
Cadence Integrators Toolkit Database Reference

**Product Version IC23.1
June 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

<u>Introduction to CDBA</u>	83
<u>Integrating Tools into a Framework</u>	84
<u>Benefits of CDBA</u>	84
<u>Database Types</u>	85
<u>dbGetDatabaseType</u>	85
<u>Supplementing CDBA Functions</u>	86

2

<u>Getting Started with CDBA</u>	87
<u>CDBA Described</u>	88
<u>Example, Include, and Library Files to Use</u>	89
<u>Finding OpenAccess Shared Libraries</u>	89
<u>Generator Functions</u>	89
<u>Sequence of Actions Used</u>	89
<u>Relationships Represented</u>	90
<u>Arguments in Generator Functions</u>	90
<u>Naming Conventions for Generator Functions</u>	91
<u>The Generator State</u>	92
<u>Starting the Generator</u>	92
<u>Generator Loop</u>	92
<u>Stopping the Generator</u>	93
<u>Get Functions</u>	93
<u>Producer and Consumer Functions</u>	94
<u>CDBA Naming Conventions</u>	95
<u>Components of Function Names</u>	95
<u>Components of Function Names</u>	96
<u>Case Used in CDBA Names</u>	98
<u>Using Abbreviations in Function Names</u>	98
<u>Naming Conventions for Function Arguments</u>	98
<u>Conventions for Prototype Functions</u>	100
<u>Return Value Conventions</u>	100

Cadence Integrators Toolkit Database Reference

<u>Naming Conventions for Name Types</u>	100
<u>Exceptions to Naming Conventions</u>	101
<u>Basics for Writing CDBA Functions</u>	101
<u>Design Framework II Requirements</u>	101
<u>Supported Platforms and Compilers for Integrator's Toolkit (itk)</u>	102
<u>C Data Types Used in CDBA</u>	103
<u>Coding Standards for CDBA Functions</u>	104
<u>Using the Virtuoso Design Environment</u>	106
<u>ICADVM20.1 Only</u>	107
<u>Specifying Run-Time Options</u>	107
<u>General Rules for Using CDBA</u>	108
<u>Troubleshooting Your Programs</u>	108

3

<u>CDBA Database Objects</u>	111
<u>Object Identifiers</u>	112
<u>Object Types and Classes</u>	112
<u>Attributes</u>	114
<u>Mandatory Attributes</u>	114
<u>Optional Attributes</u>	114
<u>Derived Attributes</u>	114
<u>Properties</u>	115
<u>Attributes and Properties Compared</u>	115
<u>Property Bags</u>	116
<u>Storage Hierarchy Structure</u>	116
<u>Property Bag Access</u>	116
<u>Explicit I/O Functions on Property Bags</u>	117
<u>dbOpenBag</u>	117
<u>dbCloseBag</u>	117
<u>dbGetRefCount</u>	118
<u>dbPurgeBag</u>	118
<u>dbSaveBag</u>	118
<u>dbRefreshBag</u>	118
<u>dbReOpenBag</u>	119
<u>dbGetBag</u>	119

Cadence Integrators Toolkit Database Reference

<u>dbMakeBagCurrent</u>	119
<u>dbGetBagDdId</u>	119
<u>dbFindOpenBag</u>	119
<u>Information Retrieval Functions on Property Bags</u>	120
<u>dbIsBagId</u>	120
<u>dbGetBagTimeStamp</u>	120
<u>dbIsBagCurrent</u>	120
<u>dbIsBagReadable</u>	120
<u>dbIsBagWritable</u>	121
<u>Relationships among Objects</u>	121
<u>Mandatory Relationships</u>	122
<u>Optional Relationships</u>	122
<u>Direct Relationships</u>	122
<u>Derived Relationships</u>	123
<u>One-to-One and Many-to-One Relationships</u>	123
<u>One-to-Many Relationships</u>	123
<u>Relationships Between Logical and Physical Objects</u>	123
<u>Objects and IDs</u>	123
<u>Conversions between IDs</u>	124
<u>Functions to Retrieve Object and ID Information</u>	124
<u>dbGetType</u>	124
<u>dbIsIdEqual</u>	124
<u>dbIsId</u>	124
<u>dbIsCellViewId</u>	125
<u>dbIsFigId</u>	125
<u>dbIsShapeId</u>	125
<u>dbIsAnyInstId</u>	125

4

CDBA Representation of Elements 127

Database Units and User Units 128

Scale Factors 128

Cadence Integrators Toolkit Database Reference

<u>User Unit Type Attributes</u>	128
<u>Functions for Database and User Unit Information</u>	129
<u>dbGetCellViewDBUPerUU</u>	129
<u>dbSetCellViewDBUPerUU</u>	130
<u>dbGetCellViewUserUnitName</u>	130
<u>dbSetCellViewUserUnitName</u>	130
<u>Functions to Convert between Database and User Units</u>	130
<u>dbDBToUserCoord</u>	131
<u>dbDBToUserPoint</u>	131
<u>dbDBToUserBBox</u>	131
<u>dbUserToDBCCoord</u>	131
<u>dbUserToDBPoint</u>	132
<u>dbUserToDBBBox</u>	132
<u>Coordinates</u>	132
<u>Points</u>	132
<u>CDBA Representation of a Point</u>	133
<u>Point Arrays</u>	134
<u>dbCopyPointArray</u>	134
<u>dbFreePointArray</u>	134
<u>Bounding Boxes</u>	134
<u>CDBA Representation of a Bounding Box</u>	135
<u>Functions and Macros for Bounding Boxes</u>	135
<u>dbLeft</u>	135
<u>dbBottom</u>	135
<u>dbRight</u>	135
<u>dbTop</u>	136
<u>dbFillBBox</u>	136
<u>dbValidBBox</u>	136
<u>dbFixBBox</u>	137
<u>dbCopyBBox</u>	137
<u>dbBBoxIntersect</u>	137
<u>dbUnionBBox</u>	137
<u>dbUnionBBox2</u>	138
<u>dbIntersectBBox</u>	138
<u>dbIntersectBBox2</u>	138
<u>User-Defined Coordinates for Bounding Boxes</u>	139

Cadence Integrators Toolkit Database Reference

<u>Functions and Macros for User Coordinates</u>	139
<u>dbUserLeft</u>	139
<u>dbUserBottom</u>	139
<u>dbUserRight</u>	139
<u>dbUserTop</u>	140
<u>dbFillUserBBox</u>	140
<u>dbFixUserBBox</u>	140
<u>Angles</u>	140
<u>CDBA Representation of an Angle</u>	141
<u>Functions to Convert between Degrees and Radians</u>	141
<u>dbDegreeToRadian</u>	141
<u>dbRadianToDegree</u>	141
<u>Shape Transformation</u>	141
<u>CDBA Representation of Shape Transformation</u>	141
<u>Functions to Transform Shapes</u>	142
<u>dbInvertTransform</u>	142
<u>dbConcatTransform</u>	142
<u>dbTransformPoint</u>	142
<u>dbTransformPointArray</u>	143
<u>dbTransformBBox</u>	143
<u>dbTransformFigRec</u>	143
<u>Distance, Direction, and Time</u>	143
<u>dbConvertTimeString</u>	144
<u>dbGetCreateTime</u>	144

5

<u>CDBA Design Management Functions</u>	145
<u>Design Data Management System</u>	146
<u>Design Library</u>	146
<u>Startup File Locator</u>	147
<u>Library List</u>	148
<u>Functions to Generate the Library List</u>	148

Cadence Integrators Toolkit Database Reference

<u>Function to Identify a 5x Library</u>	148
<u>cds.lib and lib.defs Files</u>	148
<u>Library List Generator Functions</u>	148
<u>ddStartGenLibListLib</u>	149
<u>ddGenLibListLib</u>	149
<u>ddStopGen</u>	149
<u>Function to Identify a 5X Library</u>	149
<u>ddLibs5X</u>	149
<u>Data Organization</u>	150
<u>Object Information Retrieval Functions</u>	150
<u>ddlId</u>	150
<u>ddGetObjName</u>	151
<u>ddDeleteObj</u>	151
<u>ddGetObj</u>	152
<u>ddGetObjLib</u>	156
<u>ddGetObjType</u>	156
<u>ddGetTypeName</u>	157
<u>ddGetMasterSet</u>	157
<u>ddStartGenObjRel</u>	158
<u>ddGenObjRel</u>	158
<u>ddStopGen</u>	158
<u>File Location for Reading and Writing</u>	159
<u>ddGetObjReadPath</u>	159
<u>ddGetObjWritePath</u>	159
<u>Synchronization and Access Protection</u>	159
<u>File Checkout and Checkin</u>	160
<u>ddNeedCheckout</u>	160
<u>ddCheckout</u>	160
<u>ddCheckin</u>	161

6

<u>CDBA Input and Output Functions</u>	163
<u>UNIX Directory Paths</u>	164
<u>Routines Returning Directory Paths</u>	164
<u>Initialization and Exit Functions</u>	165

Cadence Integrators Toolkit Database Reference

<u>dbInit</u>	165
<u>dbExit</u>	165
<u>illInitClients</u>	166
<u>Input and Output Functions</u>	167
<u>Libraries</u>	167
<u>dbCreateLib</u>	167
<u>dbGetLibName</u>	167
<u>Cellviews</u>	168
<u>dbOpenCellViewByType</u>	168
<u>dbReOpen</u>	171
<u>dbOpenDerivedCellView</u>	172
<u>dbClose</u>	172
<u>dbFindOpenCellView</u>	172
<u>dbGetCellView</u>	172
<u>dbPurge</u>	173
<u>dbCheck</u>	173
<u>dbSave</u>	173
<u>dbSaveAll</u>	173
<u>dbRegSaveTrigger</u>	174
<u>dbUnregSaveTrigger</u>	174
<u>dbSearchCDF</u>	174
<u>dbStartGenCellView</u>	174
<u>dbGenCellView</u>	175
<u>dbGetLibDataCompressionLevel</u>	175
<u>dbSetLibDataCompressionLevel</u>	175
<u>dbUnsetLibDataCompressionLevel</u>	175
<u>dbRefreshLibDataCompressionLevel</u>	176
<u>dbIsNonSKILLPcell</u>	176
<u>dbErrorFlushWarn</u>	177
<u>dbHasAutoSavedFile</u>	178
<u>dbHasPanicFile</u>	179
<u>dbRestoreAndOpenAutoSavedFile</u>	180
<u>dbRestoreAndOpenPanicFile</u>	181

Cadence Integrators Toolkit Database Reference

<u>Hierarchy Traversal Functions</u>	182
<u>dbOpenHier</u>	182
<u>dbOpenAllMaster</u>	182
<u>Operations on Cellviews</u>	183
<u>Cellview Attributes Functions</u>	183
<u>dbGetCellViewLibId</u>	183
<u>dbGetCellId</u>	183
<u>dbGetViewId</u>	183
<u>dbGetCellViewView</u>	183
<u>dbGetCellViewCellName</u>	184
<u>dbGetCellViewViewName</u>	184
<u>dbGetCellViewMode</u>	184
<u>dbGetCellViewFullDirPath</u>	184
<u>dbGetCellViewBBox</u>	185
<u>dbGetCellViewCellType</u>	186
<u>Cellview Voltage Range Functions</u>	188
<u>dbGetCellViewNetVoltageRange</u>	188
<u>dbIsCellViewNetVoltageRangeSet</u>	188
<u>dbSetCellViewNetVoltageRange</u>	189
<u>dbUnsetCellViewNetVoltageRange</u>	190
<u>Predicate Function for Cellviews</u>	190
<u>dbIsCellViewModified</u>	191
<u>Cellview Attributes of Purpose, Type, and Status</u>	191
<u>Cellview Type and Status Functions</u>	193
<u>dbGetCellViewType</u>	194
<u>dbGetCellViewStatus</u>	194
<u>dbSetCellViewStatus</u>	194
<u>dbIsCellViewPhysicalOnly</u>	195
<u>dbUnsetCellViewPhysicalOnly</u>	196
<u>Counter Functions</u>	197
<u>dbIsConnCurrent</u>	197
<u>dbDisableCellViewCounter</u>	198
<u>dbGetCellViewCounter</u>	199
<u>dbEnableCellViewCounter</u>	200
<u>dbSetConnCurrent</u>	201
<u>dbGetFileTimeStamp</u>	202

Cadence Integrators Toolkit Database Reference

<u>dbSetInstHeadersCounter</u>	203
<u>Database I/O Functions</u>	204
<u>dbRefreshCellView</u>	204
<u>dbWriteCellView</u>	204
<u>dbCopyCellView</u>	204
 7	
<u>CDBA Figure Functions</u>	207
<u>Figure Class Hierarchy</u>	209
<u>Operations on Figures</u>	209
<u>Retrieval and Traversal Operations</u>	209
<u>dbIsFigId</u>	210
<u>dbGetFigPurpose</u>	210
<u>dbSetFigPurpose</u>	210
<u>dbAddFigToNet</u>	210
<u>dbSubFigFromNet</u>	210
<u>dbSubFigFromPin</u>	211
<u>dbGetFigNet</u>	211
<u>dbGetFigPin</u>	211
<u>dbGetFigBBoxf</u>	211
<u>dbStartGenFig and dbGenFig</u>	212
<u>dbStartGenNonChildFig and dbGenNonChildFig</u>	212
<u>Creation and Deletion Operations</u>	213
<u>dbDeleteFig</u>	213
<u>Move and Copy Operations</u>	213
<u>dbMoveFig</u>	213
<u>dbCopyFig</u>	214
<u>Blockages</u>	215
<u>About Blockages</u>	215
<u>dbIsBlockageId</u>	216
<u>dbDeleteBlockage</u>	217
<u>dbIsBlockageOrthogonal</u>	218

Cadence Integrators Toolkit Database Reference

<u>dbIsBlockageRectangle</u>	219
<u>dbIsBlockagePushedDown</u>	220
<u>dbSetBlockagePushedDown</u>	221
<u>dbGetBlockageType</u>	222
<u>dbLayerBlockageHasEffectiveWidth</u>	223
<u>dbGetLayerBlockageEffectiveWidth</u>	224
<u>dbGetBlockageDensity</u>	225
<u>dbSetBlockageDensity</u>	226
<u>dbGetBlockageOwner</u>	227
<u>dbSetBlockageOwner</u>	228
<u>dbGetBlockage</u>	229
<u>dbGetBlockageArea</u>	230
<u>dbIsAreaBlockageId</u>	231
<u>dbCreateAreaBlockage</u>	232
<u>dbSetAreaBlockagePoint</u>	233
<u>dbTransformAreaBlockage</u>	234
<u>dbIsLayerBlockageId</u>	235
<u>dbCreateLayerBlockage</u>	236
<u>dbGetLayerBlockageLayerNum</u>	237
<u>dbSetLayerBlockageLayerNum</u>	238
<u>dbGetLayerBlockageLayerHeader</u>	239
<u>dbSetLayerBlockagePoint</u>	240
<u>dbTransformLayerBlockage</u>	241
<u>dbStartGenBlockage</u>	242
<u>dbGenBlockage</u>	243
<u>dbStartGenBlockageToPoint</u>	244
<u>dbGenBlockageToPoint</u>	245
<u>Area Halos</u>	246
<u>dbIsAreaHaloid</u>	247
<u>dbCreateAreaHalo</u>	248
<u>dbDeleteAreaHalo</u>	249
<u>dbFindAreaHalo</u>	250
<u>dbGetAreaHaloOffsets</u>	251
<u>dbSetAreaHaloOffsets</u>	252
<u>dbStartGenClusterToBlockage</u>	253
<u>dbGenClusterToBlockage</u>	254

Cadence Integrators Toolkit Database Reference

<u>dbStartGenInstToBlockage</u>	255
<u>dbGenInstToBlockage</u>	256
<u>dbStartGenPRBoundaryToBlockage</u>	257
<u>dbGenPRBoundaryToBlockage</u>	258
<u>dbCoverObstructionId</u>	259
<u>dbDeleteCoverObstruction</u>	259
<u>dbPRBoundaryId</u>	259
<u>dbGetCoverObstructionPoints</u>	259
<u>dbGetCoverObstructionMaskRange</u>	260
<u>dbStartGenCoverObstructionToLayerHeader</u>	260
<u>dbGenCoverObstructionToLayerHeader</u>	260
<u>dbGetCoverObstructionLayers</u>	261
<u>dbGetCoverObstructionAllowPGNet</u>	261
<u>dbSetCoverObstructionAllowPGNet</u>	261
<u>dbIsCoverObstructionId</u>	262
<u>dbPRBoundaryHasCoverObstruction</u>	262
<u>dbGetPRBoundaryCoverObstruction</u>	262
<u>Layer Halos</u>	263
<u>About Layer Halo Objects</u>	263
<u>dbIsLayerHaloid</u>	265
<u>dbCreateLayerHalo</u>	266
<u>dbGetLayerHaloLayerNum</u>	267
<u>dbSetLayerHaloLayerNum</u>	268
<u>dbGetLayerHaloLayerHeader</u>	269
<u>dbGetLayerHaloOffsets</u>	270
<u>dbSetLayerHaloOffsets</u>	271
<u>Boundaries</u>	272
<u>About Boundaries</u>	272
<u>dbIsBoundaryId</u>	273
<u>dbIsAreaBoundaryId</u>	274
<u>dbIsClusterBoundaryId</u>	275
<u>dbIsPRBoundaryId</u>	276
<u>dbIsSnapBoundaryId</u>	277

Cadence Integrators Toolkit Database Reference

<u>dbCreateAreaBoundary</u>	278
<u>dbCreateClusterBoundary</u>	279
<u>dbCreatePRBoundary</u>	280
<u>dbCreateSnapBoundary</u>	281
<u>dbDeleteBoundary</u>	282
<u>dbFindAreaBoundaryByName</u>	283
<u>dbFindClusterBoundaryByName</u>	284
<u>dbGetAreaBoundary</u>	285
<u>dbSetAreaBoundary</u>	286
<u>dbGetPRBoundary</u>	287
<u>dbGetPRBoundaryCoreBoxSpec</u>	288
<u>dbSetPRBoundaryCoreBoxSpec</u>	289
<u>dbRemovePRBoundaryCoreBoxSpec</u>	290
<u>dbGetPRBoundaryIOBox</u>	291
<u>dbSetPRBoundaryIOBox</u>	292
<u>dbRemovePRBoundaryIOBox</u>	293
<u>dbStartGenPRBoundaryToBlockage</u>	294
<u>dbGetCoreBoxSpecNumRows</u>	295
<u>dbGetCoreBoxSpecRowBBox</u>	296
<u>dbGetSnapBoundary</u>	297
<u>dbTransformBoundary</u>	298
<u>dbGetClusterBoundaryName</u>	299
<u>dbSetClusterBoundaryName</u>	300
<u>dbGetClusterBoundaryCluster</u>	301
<u>dbIsBoundaryOrthogonal</u>	302
<u>dbIsBoundaryRectangle</u>	303
<u>dbGetBoundaryArea</u>	304
<u>dbGetBoundaryEdge</u>	305
<u>dbSetBoundaryEdge</u>	306
<u>dbStartGenAreaBoundary</u>	307
<u>dbGenAreaBoundary</u>	308
<u>dbStartGenClusterToBoundary</u>	309
<u>dbGenClusterToBoundary</u>	310
<u>dbStartGenBoundaryToPoint</u>	311
<u>dbGenBoundaryToPoint</u>	312
<u>Clusters</u>	313

Cadence Integrators Toolkit Database Reference

<u>dbStartGenCluster</u>	314
<u>dbGenCluster</u>	315
<u>LayerHeaders</u>	316
<u>dbIsLayerHeaderId</u>	317
<u>dbFindLayerHeader</u>	318
<u>dbGetLayerHeaderLayerNum</u>	319
<u>dbIsLayerHeaderBound</u>	320
<u>dbGetLayerSteinerBBox</u>	321
<u>dbGetLayerGuideBBox</u>	322
<u>dbGetLayerBlockageBBox</u>	323
<u>dbGetLPPHeaderLayerHeader</u>	324
<u>dbStartGenLayerHeader</u>	325
<u>dbGenLayerHeader</u>	326
<u>dbStartGenLayerToSteiner</u>	327
<u>dbGenLayerToSteiner</u>	328
<u>dbStartGenLayerToGuide</u>	329
<u>dbGenLayerToGuide</u>	330
<u>dbStartGenLayerToBlockage</u>	331
<u>dbGenLayerToBlockage</u>	332
<u>dbStartGenLayerHeaderToLPPHeader</u>	333
<u>dbGenLayerHeaderToLPPHeader</u>	334
<u>Markers</u>	335
<u>About Markers</u>	335
<u>dbIsMarkerId</u>	336
<u>dbCreateMarker</u>	337
<u>dbDeleteMarker</u>	339
<u>dbTransformMarker</u>	340
<u>dbAddObjectToMarker</u>	341
<u>dbRemoveObjectFromMarker</u>	342
<u>dbIsMarkerVisible</u>	343
<u>dbSetMarkerVisible</u>	344
<u>dbIsMarkerClosed</u>	345
<u>dbSetMarkerClosed</u>	346

Cadence Integrators Toolkit Database Reference

<u>dbGetMarkerSeverity</u>	347
<u>dbSetMarkerSeverity</u>	348
<u>dbMarkerSeverityToString</u>	349
<u>dbStringToMarkerSeverity</u>	350
<u>dbGetMarkerMsg</u>	351
<u>dbSetMarkerMsg</u>	352
<u>dbGetMarkerShortMsg</u>	353
<u>dbSetMarkerShortMsg</u>	354
<u>dbGetMarkerTool</u>	355
<u>dbSetMarkerTool</u>	356
<u>dbGetMarkerDeleteWhen</u>	357
<u>dbSetMarkerDeleteWhen</u>	358
<u>dbGetMarker</u>	359
<u>dbSetMarkerPoint</u>	360
<u>dbStartGenMarker</u>	361
<u>dbGenMarker</u>	362
<u>dbStartGenMarkerToPoint</u>	363
<u>dbGenMarkerToPoint</u>	364
<u>dbStartGenMarkerToObject</u>	365
<u>dbGenMarkerToObject</u>	366
<u>dbStartGenObjectToMarker</u>	367
<u>dbGenObjectToMarker</u>	368
<u>RowHeaders</u>	369
<u>About rowHeaders</u>	369
<u>dbIsRowHeaderId</u>	370
<u>dbFindRowHeader</u>	371
<u>dbGetRowHeaderSiteDef</u>	372
<u>dbGetRowHeaderSiteDefName</u>	373
<u>dbGetRowHeaderSiteDefWidth</u>	374
<u>dbGetRowHeaderSiteDefHeight</u>	375
<u>dbGetRowHeaderBBox</u>	376
<u>dbGetRowHeaderNumRows</u>	377
<u>dbIsRowHeaderBound</u>	378
<u>dbStartGenRowHeader</u>	379
<u>dbGenRowHeader</u>	380
<u>EEQ Masters</u>	381

Cadence Integrators Toolkit Database Reference

<u>dbGetCellViewEEQMaster</u>	381
<u>dbSetCellViewEEQMaster</u>	381
<u>Steiners</u>	382
<u>dbIsSteinerId</u>	383
<u>dbDeleteSteiner</u>	384
<u>dbCreateSteiner</u>	385
<u>dbGetSteinerBBox</u>	386
<u>dbSetSteinerBBox</u>	387
<u>dbGetSteinerLayerHeader</u>	388
<u>dbGetSteinerLayerNum</u>	389
<u>dbSetSteinerLayerNum</u>	390
<u>dbStartGenSteiner</u>	391
<u>dbGenSteiner</u>	392
<u>dbStartGenSteinerToRoute</u>	393
<u>dbGenSteinerToRoute</u>	394
<u>dbCreateTapSteiner</u>	395
<u>dbIsTapSteiner</u>	396
<u>Track Patterns</u>	397
<u>About trackPatterns</u>	397
<u>dbIsTrackPatternId</u>	398
<u>dbCreateTrackPattern</u>	399
<u>dbDeleteTrackPattern</u>	400
<u>dbGetTrackPatternIsHorizontal</u>	401
<u>dbGetTrackPatternStart</u>	402
<u>dbSetTrackPatternStart</u>	403
<u>dbGetTrackPatternNumTracks</u>	404
<u>dbSetTrackPatternNumTracks</u>	405
<u>dbGetTrackPatternSpace</u>	406
<u>dbSetTrackPatternSpace</u>	407
<u>dbGetTrackPatternRoutingLayer</u>	408
<u>dbSetTrackPatternRoutingLayer</u>	409
<u>dbStartGenTrackPattern</u>	410
<u>dbGenTrackPattern</u>	411

Cadence Integrators Toolkit Database Reference

<u>Shape Figure Functions</u>	412
<u>Hierarchy of Shapes</u>	412
<u>Operations on Generic Shapes</u>	412
<u>dbIsShapeld</u>	412
<u>dbStartGenShape</u>	413
<u>dbGenShape</u>	413
<u>dbStartGenNonChildShape</u>	413
<u>dbGenNonChildShape</u>	413
<u>dbStartGenLPToShape</u>	414
<u>dbGenLPToShape</u>	414
<u>dbStartGenLPToPinShape</u>	414
<u>dbGenLPToPinShape</u>	414
<u>dbStartGenLPToNonChildShape</u>	415
<u>dbGenLPToNonChildShape</u>	415
<u>dbDeleteShape</u>	415
<u>dbMoveShape</u>	415
<u>dbCopyShape</u>	416
<u>dbGetShapeLP</u>	417
<u>dbGetShapeBBox</u>	417
<u>dbSetShapeLayer</u>	417
<u>dbSetShapePurpose</u>	418
<u>dbSetViaShapePurpose</u>	418
<u>dbSetShapeLP</u>	418
<u>dbHasShape</u>	418
<u>dbLPHasShape</u>	419
<u>dbGetShapeCellView</u>	419
<u>Rectangles</u>	420
<u>dbCreateRect</u>	420
<u>dbSetRectBBox</u>	420
<u>dbDeleteRect</u>	420
<u>dbGetRectLP</u>	420
<u>dbGetRectBBox</u>	421
<u>dbConvertRectToPolygon</u>	421
<u>Polygons</u>	422
<u>dbCreatePolygon</u>	422
<u>dbSetPolygonPoint</u>	422

Cadence Integrators Toolkit Database Reference

<u>dbDeletePolygon</u>	422
<u>dbGetPolygonLP</u>	422
<u>dbGetPolygon</u>	423
<u>dbGetPolygonBBox</u>	423
<u>dbStartGenPolygonToPoint</u>	423
<u>dbGenPolygonToPoint</u>	423
<u>dbConvertPolygonToRect</u>	424
<u>Lines</u>	425
<u>dbCreateLine</u>	425
<u>dbSetLinePoint</u>	425
<u>dbDeleteLine</u>	425
<u>dbGetLineLP</u>	425
<u>dbGetLine</u>	426
<u>dbGetLineBBox</u>	426
<u>dbStartGenLineToPoint</u>	426
<u>dbGenLineToPoint</u>	426
<u>Paths</u>	428
<u>dbCreatePath</u>	428
<u>dbSetPathPoint</u>	428
<u>dbSetPathExt</u>	429
<u>dbSetPathStyle</u>	429
<u>dbSetPathWidth</u>	429
<u>dbDeletePath</u>	429
<u>dbGetPathLP</u>	430
<u>dbGetPath</u>	430
<u>dbGetPathBBox</u>	430
<u>dbGetPathExt</u>	430
<u>dbGetPathWidth</u>	431
<u>dbGetPathStyle</u>	431
<u>dbStartGenPathToPoint</u>	431
<u>dbGenPathToPoint</u>	431
<u>dbConvertPathToPolygon</u>	432
<u>dbConvertLineToPath</u>	432

Cadence Integrators Toolkit Database Reference

<u>Labels</u>	433
<u>Drafting</u>	433
<u>dbCreateLabel</u>	433
<u>dbSetLabelType</u>	433
<u>dbSetLabelFontStyle</u>	434
<u>dbSetLabelOverbar</u>	434
<u>dbSetLabelDrafting</u>	434
<u>dbSetLabelOrigin</u>	434
<u>dbSetLabelTheLabel</u>	435
<u>dbSetLabelJustify</u>	435
<u>dbSetLabelOrient</u>	435
<u>dbSetLabelHeight</u>	435
<u>dbDeleteLabel</u>	435
<u>dbGetLabel</u>	436
<u>dbGetLabelOrigin</u>	436
<u>dbGetLabelLP</u>	436
<u>dbGetLabelBBox</u>	436
<u>dbGetLabelType</u>	436
<u>dbGetLabelOverbar</u>	437
<u>dbGetLabelDrafting</u>	437
<u>Font Height</u>	437
<u>Font Style</u>	437
<u>Justify</u>	438
<u>Label Type</u>	438
<u>Origin</u>	439
<u>Overbar</u>	439
<u>Rotation</u>	439
<u>Ellipses</u>	440
<u>dbCreateEllipse</u>	440
<u>dbSetEllipseBBox</u>	440
<u>dbDeleteEllipse</u>	440
<u>dbGetEllipseLP</u>	440
<u>dbGetEllipseBBox</u>	441
<u>dbConvertEllipseToPolygon</u>	441
<u>Arcs</u>	442
<u>dbCreateArc</u>	442

Cadence Integrators Toolkit Database Reference

<u>dbSetArc</u>	442
<u>dbCreateArcByThreePoints</u>	442
<u>dbSetArcPoint</u>	443
<u>dbDeleteArc</u>	443
<u>dbGetArcLP</u>	443
<u>dbGetArc</u>	443
<u>dbGetArcBBox</u>	444
<u>dbGetArcBothBBox</u>	444
<u>dbConvertArcToLine</u>	444
<u>Donuts</u>	445
<u>dbCreateDonut</u>	445
<u>dbSetDonutCenter</u>	445
<u>dbSetDonutRadius</u>	445
<u>dbDeleteDonut</u>	445
<u>dbGetDonutLP</u>	446
<u>dbGetDonutBBox</u>	446
<u>dbGetDonutBothBBox</u>	446
<u>dbConvertDonutToPolygon</u>	446
<u>Dots</u>	448
<u>dbCreateDot</u>	448
<u>dbSetDot</u>	448
<u>dbSetDotSize</u>	448
<u>dbDeleteDot</u>	448
<u>dbGetDotLP</u>	449
<u>dbGetDot</u>	449
<u>Shape Route Status</u>	450
<u>dbGetShapeRouteStatus</u>	450
<u>dbSetShapeRouteStatus</u>	451
<u>Text Display Objects</u>	452
<u>Text Display Types</u>	452
<u>The Text Display Function</u>	452
<u>dbTextDisplayId</u>	452
<u>Macros for Display Flags</u>	453

Cadence Integrators Toolkit Database Reference

<u>dbmSetDisplayFlagsAll</u>	453
<u>dbmUnsetDisplayFlagsAll</u>	453
<u>dbmSetDisplayFlagsOverbar</u>	454
<u>dbmUnsetDisplayFlagsOverbar</u>	454
<u>dbmSetDisplayFlagsDrafting</u>	454
<u>dbmUnsetDisplayFlagsDrafting</u>	454
<u>dbmSetDisplayFlagsFontStyle</u>	454
<u>dbmUnsetDisplayFlagsFontStyle</u>	455
<u>dbmSetDisplayFlagsHeight</u>	455
<u>dbmUnsetDisplayFlagsHeight</u>	455
<u>dbmSetDisplayFlagsJustify</u>	455
<u>dbmUnsetDisplayFlagsJustify</u>	455
<u>dbmSetDisplayFlagsOrient</u>	456
<u>dbmUnsetDisplayFlagsOrient</u>	456
<u>dbmSetDisplayFlagsOrigin</u>	456
<u>dbmUnsetDisplayFlagsOrigin</u>	456
<u>dbmSetDisplayFlagsVisible</u>	456
<u>dbmUnsetDisplayFlagsVisible</u>	457
<u>Structure and Functions for Text Display Records</u>	457
<u>dbSetTextDisplayOverbar</u>	458
<u>dbSetTextDisplayFontStyle</u>	458
<u>dbSetTextDisplayDrafting</u>	458
<u>dbSetTextDisplayOrigin</u>	458
<u>dbSetTextDisplayJustify</u>	458
<u>dbSetTextDisplayOrient</u>	459
<u>dbSetTextDisplayHeight</u>	459
<u>dbDeleteTextDisplay</u>	459
<u>dbGetTextDisplay</u>	459
<u>dbGetTextDisplayOrigin</u>	460
<u>dbGetTextDisplayLP</u>	460
<u>dbGetTextDisplayBBox</u>	460
<u>dbGetTextDisplayAssociate</u>	460
<u>dbGetTextDisplayOwner</u>	461
<u>dbAssociateHasTextDisplay</u>	461
<u>dbObjectHasTextDisplay</u>	461
<u>dbGetAssociateTextDisplay</u>	461

Cadence Integrators Toolkit Database Reference

<u>dbStartGenTextDisplay</u>	462
<u>dbGenTextDisplay</u>	462
<u>dbSetTextDisplayVisible</u>	462
<u>dbSetTextDisplayNameVisible</u>	462
<u>dbSetTextDisplayValueVisible</u>	462
<u>Instance and Hierarchy Functions</u>	463
<u>Instances</u>	463
<u>Iterated Instances</u>	463
<u>Instance Name Syntax</u>	463
<u>Instance Headers</u>	464
<u>Binding of Instances and Instance Headers</u>	464
<u>Automatic Actions on Instance Headers and Cellviews</u>	465
<u>Instance Creation and Modification</u>	466
<u>dbCreateInst</u>	466
<u>dbCreateInstByMasterName</u>	467
<u>dbCreateBlockInst</u>	467
<u>dbCreateBlockInstByMasterName</u>	468
<u>dbCreateInstWithPurpose</u>	468
<u>dbCreateParamInst</u>	468
<u>dbSetInstPlacementStatus</u>	469
<u>dbSetInstName</u>	469
<u>dbSetInstOrigin</u>	470
<u>dbSetInstOrient</u>	470
<u>dbSetInstMag</u>	470
<u>dbSetInstTransform</u>	470
<u>dbSetInstMaster</u>	471
<u>dbSetAnyInstMaster</u>	471
<u>dbSetAnyInstNumInst</u>	471
<u>dbSetAnyInstName</u>	472
<u>dbSetAnyInstBaseName</u>	472
<u>dbAllocFlattenOptions</u>	472
<u>dbFreeFlattenOptions</u>	473
<u>dbFlattenInst</u>	474

Cadence Integrators Toolkit Database Reference

<u>dbFlattenInst2</u>	476
<u>dbCreateExcludePCellsFromFile</u>	480
<u>dbExcludePCellsContain</u>	482
<u>dbDeleteExcludePCells</u>	483
<u>Instance Deletion</u>	484
<u>dbDeleteInst</u>	484
<u>Instance Information Retrieval</u>	485
<u>General Instance Information</u>	485
<u>dbGetInst</u>	485
<u>dbGetInstTransform</u>	485
<u>dbGetInstOrigin</u>	485
<u>dbGetInstOrient</u>	485
<u>dbGetInstMag</u>	486
<u>dbGetInstName</u>	486
<u>dbGetInstNumInst</u>	486
<u>dbGetInstPlacementStatus</u>	486
<u>dbGetInstBBox</u>	487
<u>dbGetInstLibName</u>	487
<u>dbGetInstCellName</u>	487
<u>dbGetInstViewName</u>	487
<u>dbGetInstNameNumInst</u>	487
<u>dbGetAnyInstNumInst</u>	488
<u>dbGetAnyInstName</u>	488
<u>dbGetAnyInstBaseName</u>	488
<u>Instance Headers</u>	488
<u>dbGetInstInstHeader</u>	489
<u>dbGetInstHeaderMaster</u>	489
<u>dbGetAnyInstInstHeader</u>	489
<u>Instance Masters</u>	490
<u>dbGetInstMaster</u>	490
<u>dbGetAnyInstMaster</u>	490
<u>dbGetAnyInstSwitchMaster</u>	490
<u>Predicate Functions for Instances</u>	491
<u>dbIsInstHeaderBound</u>	491
<u>dbIsSubInstHeaderId</u>	491
<u>dbIsInstBound</u>	491

Cadence Integrators Toolkit Database Reference

<u>dbIsAnyInstBound</u>	491
<u>dbIsSameMaster</u>	492
<u>Generator Functions</u>	493
<u>dbStartGenInstHeader</u>	493
<u>dbGenInstHeader</u>	493
<u>dbStartGenMaster</u>	493
<u>dbGenMaster</u>	493
<u>dbStartGenInstHeaderToAnyInst</u>	494
<u>dbGenInstHeaderToAnyInst</u>	494
<u>dbStartGenInstHeaderToInst</u>	494
<u>dbGenInstHeaderToInst</u>	494
<u>dbStartGenAnyInst</u>	495
<u>dbGenAnyInst</u>	495
<u>dbStartGenInst</u>	495
<u>dbGenInst</u>	496
<u>dbStartGenMasterToInstHeader</u>	496
<u>dbGenMasterToInstHeader</u>	496
<u>Hierarchy Traversal Functions</u>	497
<u>dbProduceOverlap</u>	497
<u>usrShapeConsumer</u>	497
<u>dbProduceOverlapInst</u>	498
<u>usrInstConsumer</u>	498
<u>dbGetNeighbor</u>	499
<u>dbGetNeighborInst</u>	499
<u>Advanced Hierarchy Traversal Functions</u>	501
<u>dbHierPath Structure</u>	501
<u>dbHierInfo Structure</u>	501
<u>Functions for the HierInfo Structure</u>	505
<u>dbAllocHierInfo</u>	505
<u>dbFreeHierInfo</u>	505
<u>dbProduceOverlap2</u>	505
<u>usrConsumer</u>	506
<u>usrSwitcher</u>	507

Cadence Integrators Toolkit Database Reference

<u>mosaicSwitcher</u>	507
<u>viaSwitcher</u>	508
<u>Hierarchy Signal Traversing Functions</u>	509
<u>dbSigHierPath Structure</u>	509
<u>dbSigHierInfo Structure</u>	509
<u>Functions Using the dbSigHierInfo Structure</u>	511
<u>dbTraverseSigHier</u>	511
<u>usrConsumer</u>	511
<u>usrSwitcher</u>	512
<u>FigGroup Functions</u>	513
<u>About figGroups</u>	513
<u>dbCreateFigGroup</u>	514
<u>dbDeleteFigGroup</u>	514
<u>dbAddFigToFigGroup</u>	514
<u>dbFindFigGroupByName</u>	515
<u>dbRemoveFigFromFigGroup</u>	515
<u>dbSetFigGroupOrient</u>	515
<u>dbSetFigGroupOrigin</u>	516
<u>dbGetFigGroupOrdered</u>	516
<u>dbSetFigGroupOrdered</u>	516
<u>dbGetFigGroupName</u>	517
<u>dbGetFigGroupOrientation</u>	517
<u>dbGetFigGroupOrigin</u>	517
<u>dbGetFigGroupType</u>	518
<u>dbGetMemberFigGroup</u>	518
<u>dbStartGenFigGroup</u>	518
<u>dbGenFigGroup</u>	518
<u>dbStartGenFigGroupTold</u>	519
<u>dbGenFigGroupTold</u>	519
<u>Region Queries</u>	520
<u>dbLayerBlockageQuery</u>	523
<u>dbPlacementBlockageQuery</u>	525
<u>dbBoundaryQuery</u>	527
<u>dbGuideQuery</u>	529
<u>dbInstQuery</u>	531
<u>dbMarkerQuery</u>	533

Cadence Integrators Toolkit Database Reference

<u>dbRowQuery</u>	535
<u>dbShapeQuery</u>	537
<u>dbSteinerQuery</u>	539
<u>dbViaQuery</u>	541
<u>dbAbortQuery</u>	543
<u>dbGetQueryLayer</u>	544
<u>dbGetQueryPurpose</u>	545
<u>dbSetQueryClientData</u>	546
<u>dbGetQueryClientData</u>	547
<u>dbGetQueryTopCellView</u>	548
<u>dbGetQueryRegion</u>	549
<u>dbGetQueryCurrentTransform</u>	550
<u>dbGetQueryCurrentRegion</u>	551
<u>dbCreateQueryHierPath</u>	552
<u>dbFreeQueryHierPath</u>	553
<u>dbGetQueryHierPath</u>	554
<u>dbCopyQueryHierPath</u>	555
<u>dbCompareQueryHierPaths</u>	556
<u>dbResetQueryHierPath</u>	557
<u>dbGetQueryHierPathTransform</u>	558
<u>dbGetQueryHierPathDepth</u>	559
<u>dbGetQueryHierPathLevel</u>	560
<u>dbPushQueryHierPathLevel</u>	561
<u>dbPopQueryHierPath</u>	562
<u>Parameterized Cells</u>	563
<u>Parameterized Cells and Database Organization</u>	563
<u>Functions for Parameterized Cells</u>	564
<u>dbCellViewHasVariant</u>	564
<u>dbStartGenVariantCellView</u>	565
<u>dbGenVariantCellView</u>	565
<u>dbInstHeaderHasVariant</u>	565
<u>dbStartGenInstHeaderVariant</u>	565
<u>dbGenInstHeaderVariant</u>	565

Cadence Integrators Toolkit Database Reference

<u>dbDumpPcell</u>	566
<u>Defining and Undefined a Parameterized Cell</u>	567
<u>dbDefineParamCell</u>	567
<u>dbUndefineParamCell</u>	568
<u>Creating Parameterized Cells</u>	568
<u>Warnings about Parameterized Cells</u>	570
<u>Abutment Attributes</u>	571
<u>Abutment Attributes for Parametrized Cells</u>	571
<u>dbSetFigAbutName</u>	572
<u>dbSetFigAbutRefName</u>	573
<u>dbGetFigAbutName</u>	574
<u>dbGetFigAbutRefName</u>	575
<u>dbHasFigAbutName</u>	576
<u>dbHasFigAbutRefName</u>	577
<u>dbFindAbutFig</u>	578
<u>Mosaic Functions</u>	579
<u>Mosaic and Mosaic Instance Creation</u>	579
<u>dbCreateSimpleMosaic</u>	579
<u>dbCreateDefaultSpacingMosaic</u>	579
<u>dbCreateMosaic</u>	580
<u>dbCreateMosaicInst</u>	580
<u>Mosaic and Mosaic Instance Deletion</u>	580
<u>dbDeleteMosaic</u>	580
<u>dbDeleteMosaicInst</u>	581
<u>Mosaic Modification</u>	581
<u>dbSetRowOffsetByIndex</u>	581
<u>dbSetColumnOffsetByIndex</u>	581
<u>dbSetRowOffset</u>	582
<u>dbSetColumnOffset</u>	582
<u>dbSetUniformRowSpacing</u>	582
<u>dbSetUniformColumnSpacing</u>	582
<u>dbSetMosaicName</u>	583
<u>dbSetMosaicType</u>	583
<u>dbSetMosaicTileArray</u>	583
<u>dbSetMosaicInstMaster</u>	583
<u>dbSetMosaicSize</u>	584

Cadence Integrators Toolkit Database Reference

<u>dbSetMosaicOrigin</u>	584
<u>Mosaic and Mosaic Instance Information Retrieval</u>	584
<u>dbGetMosaicType</u>	584
<u>dbGetMosaicTileArray</u>	584
<u>dbGetMosaicOrigin</u>	585
<u>dbGetMosaicName</u>	585
<u>dbGetMosaicSize</u>	585
<u>dbGetMosaicNumInst</u>	585
<u>dbGetMosaicNumRows</u>	586
<u>dbGetMosaicNumColumns</u>	586
<u>dbGetMosaicMaxExt</u>	587
<u>dbGetMosaicMasterIcon</u>	587
<u>dbGetMosaicByName</u>	587
<u>dbGetMosaicOutline</u>	587
<u>dbGetMosaicBBox</u>	587
<u>dbGetMosaicInstMosaic</u>	588
<u>dbGetMosaicInstIcon</u>	588
<u>dbGetMosaicInstBBox</u>	588
<u>dbGetMosaicInstTransform</u>	588
<u>dbGetRowOffsetByIndex</u>	589
<u>dbGetColumnOffsetByIndex</u>	589
<u>dbGetRowOffset</u>	589
<u>dbGetColumnOffset</u>	589
<u>dbGetSimpleMosaicSpacing</u>	589
<u>dbGetUniformRowSpacing</u>	590
<u>dbGetUniformColumnSpacing</u>	590
<u>dbAllocMosaicTileArray</u>	590
<u>dbFreeMosaicTileArray</u>	590
<u>Boolean Predicate Retrieval Functions</u>	590
<u>dbIsColumnSpacingUniform</u>	591
<u>dbIsRowSpacingUniform</u>	591
<u>dbIsSimpleMosaic</u>	591
<u>dbHasMosaic</u>	591

Cadence Integrators Toolkit Database Reference

<u>Traversal and Generator Functions</u>	591
<u>dbStartGenMosaic</u>	592
<u>dbGenMosaic</u>	592
<u>dbStartGenMosaicToMosaicInst</u>	592
<u>dbGenMosaicToMosaicInst</u>	592
<u>dbStartGenMosaicToElementInfo</u>	593
<u>dbGenMosaicToElementInfo</u>	593
<u>Guides</u>	594
<u>About Guides</u>	594
<u>dbIsGuideld</u>	595
<u>dbCreateGuide</u>	596
<u>dbDeleteGuide</u>	597
<u>dbGetGuidePoints</u>	598
<u>dbSetGuidePoints</u>	599
<u>dbGetGuideRoute</u>	600
<u>dbIsGuideFollowLayers</u>	601
<u>dbSetGuideFollowLayers</u>	602
<u>dbStartGenGuide</u>	603
<u>dbGenGuide</u>	604
<u>pathSegs</u>	605
<u>About pathSeg objects</u>	605
<u>dbIsPathSegId</u>	607
<u>dbCreatePathSeg</u>	608
<u>dbDeletePathSeg</u>	610
<u>dbConvertPathSegToPolygon</u>	611
<u>dbGetPathSegPoints</u>	612
<u>dbGetPathSegBoundary</u>	613
<u>dbGetPathSegRecBoundary</u>	614
<u>dbGetPathSegStyle</u>	616
<u>dbGetPathSegRouteTopology</u>	617
<u>dbIsPathSegOrthogonal</u>	618
<u>dbSetPathSegPoints</u>	619
<u>dbSetPathSegStyle</u>	620
<u>dbSetPathSegTopology</u>	621
<u>dbGetPathSegRoute</u>	622
<u>Vias</u>	623

Cadence Integrators Toolkit Database Reference

<u>About Vias</u>	623
<u>dbIsViald</u>	625
<u>dbDeleteVia</u>	626
<u>dbGetViaViaHeader</u>	627
<u>dbGetViaViaDef</u>	628
<u>dbGetViaViaDefName</u>	629
<u>dbGetViaRoute</u>	630
<u>dbGetViaRouteStatus</u>	631
<u>dbGetViaDirection</u>	632
<u>dbSetViaRouteStatus</u>	633
<u>dbSetViaDirection</u>	634
<u>dbStartGenViaHeaderToVia</u>	635
<u>dbGenViaHeaderToVia</u>	636
<u>dbIsCustomViald</u>	637
<u>dbCreateCustomVia</u>	638
<u>dbGetCustomViaParams</u>	639
<u>dbSetCustomViaParams</u>	640
<u>dbIsStdViald</u>	641
<u>dbCreateStdVia</u>	642
<u>dbGetStdViaViaParam</u>	645
<u>dbSetStdViaParam</u>	646
<u>Routes</u>	647
<u>About Routes</u>	647
<u>dbIsRouteId</u>	648
<u>dbCreateRoute</u>	649
<u>dbDeleteRoute</u>	650
<u>dbGetRouteObjects</u>	651
<u>dbGetRouteNumObjects</u>	652
<u>dbGetRouteStatus</u>	653
<u>dbGetRouteStartConn</u>	654
<u>dbGetRouteEndConn</u>	655
<u>dbIsRouteGlobal</u>	656
<u>dbIsRouteHasGuide</u>	657

Cadence Integrators Toolkit Database Reference

<u>dbIsRouteContiguous</u>	658
<u>dbSetRouteStatus</u>	659
<u>dbSetRouteObjects</u>	660
<u>dbSetRouteGlobal</u>	661
<u>dbSetRouteStartConn</u>	662
<u>dbSetRouteEndConn</u>	663
<u>dbUnsetRouteStartConn</u>	664
<u>dbUnsetRouteEndConn</u>	665
<u>dbGetNumRoutes</u>	666
<u>dbStartGenRoute</u>	667
<u>dbGenRoute</u>	668
<u>dbStartGenNetToRoute</u>	669
<u>dbGenNetToRoute</u>	670
<u>dbStartGenShapeToConnRoute</u>	671
<u>dbGenShapeToConnRoute</u>	672
<u>dbStartGenPinToConnRoute</u>	673
<u>dbGenPinToConnRoute</u>	674
<u>dbStartGenTermToConnRoute</u>	675
<u>dbGenTermToConnRoute</u>	676
<u>dbStartGenInstTermToConnRoute</u>	677
<u>dbGenInstTermToConnRoute</u>	678
<u>CDBA Connectivity Functions</u>	679
<u>Object Classes and Their Members</u>	680
<u>Figures</u>	681
<u>Connections in Cellviews</u>	681
<u>Any Instance</u>	681
<u>Instances</u>	682
<u>Functions Acting on Instances</u>	682
<u>dbGetMemInstName</u>	682
<u>dbStartGenAnyInstToConn</u>	682
<u>dbGenAnyInstToConn</u>	683
<u>dbStartGenAnyInstToInstTerm</u>	683
<u>dbGenAnyInstToInstTerm</u>	683
<u>Nets</u>	684
<u>Nets and Signals</u>	684
<u>Net Hierarchy</u>	684

Cadence Integrators Toolkit Database Reference

<u>Net Names</u>	685
<u>Connection Status</u>	685
<u>Functions Acting on Nets</u>	685
<u>dbCreateNet</u>	686
<u>dbCreateNamedSubNet</u>	687
<u>dbMakeNet</u>	688
<u>dbCreateUniqueNamedNet</u>	688
<u>dbDeleteNet</u>	688
<u>dbDeleteEmptyNet</u>	689
<u>dbGetNetName</u>	689
<u>dbGetNetNumBit</u>	689
<u>dbGetMemNetName</u>	689
<u>dbGetMemNetSig</u>	689
<u>dbGetNetTerm</u>	690
<u>dbGetNetParent</u>	690
<u>dbGetNetRootNet</u>	690
<u>dbIsNetVoltageRangeSet</u>	691
<u>dbGetNetVoltageRangeOnNet</u>	691
<u>dbIsNetVoltageRangeSetOnNet</u>	692
<u>dbGetNetVoltageRangeSource</u>	693
<u>dbSetNetVoltageRangeSource</u>	693
<u>dbUnsetNetVoltageRangeSource</u>	694
<u>dbIsNetOptical</u>	695
<u>dbMoveChildNet</u>	695
<u>dbMergeNet</u>	695
<u>dbNetHasGlobal</u>	696
<u>dbNetHasFig</u>	696
<u>dbNetHasTerm</u>	696
<u>dbNetHasInstTerm</u>	696
<u>dbNetHasPin</u>	697
<u>dbNetHasChild</u>	697
<u>dbGetNetPriority</u>	697
<u>dbSetNetPriority</u>	697

Cadence Integrators Toolkit Database Reference

<u>dbGetNetConnStatus</u>	697
<u>dbSetNetConnStatus</u>	698
<u>dbGetNetPowerDomain</u>	698
<u>dbSetNetPowerDomain</u>	699
<u>dbStartGenNetToSig</u>	699
<u>dbGenNetToSig</u>	700
<u>dbStartGenNetToFig</u>	700
<u>dbGenNetToFig</u>	700
<u>dbStartGenNetToInstTerm</u>	700
<u>dbGenNetToInstTerm</u>	701
<u>dbStartGenNetToPin</u>	701
<u>dbGenNetToPin</u>	701
<u>dbStartGenNetToAllInstTerm</u>	701
<u>dbGenNetToAllInstTerm</u>	701
<u>dbStartGenNetToChild</u>	702
<u>dbGenNetToChild</u>	702
<u>dbStartGenNetToAllChild</u>	702
<u>dbGenNetToAllChild</u>	702
<u>dbGetNetRoutePattern</u>	703
<u>dbSetNetRoutePattern</u>	704
<u>Signals</u>	705
<u>Hierarchy for Signals</u>	705
<u>Signal Types</u>	705
<u>Functions Acting on Signals</u>	706
<u>dbGetSigName</u>	706
<u>dbSetSigName</u>	706
<u>dbIsSigGlobal</u>	706
<u>dbIsSigOptical</u>	706
<u>dbSetSigGlobal</u>	707
<u>dbResetSigGlobal</u>	707
<u>dbStartGenSigToAlias</u>	707
<u>dbGenSigToAlias</u>	708
<u>dbGetSigType</u>	708
<u>dbSetSigType</u>	708
<u>dbStartGenSigToMemNet</u>	709
<u>dbGenSigToMemNet</u>	709

Cadence Integrators Toolkit Database Reference

<u>dbStartGenSigToMemTerm</u>	709
<u>dbGenSigToMemTerm</u>	709
<u>dbStartGenSigToMemInstTerm</u>	709
<u>dbGenSigToMemInstTerm</u>	710
<u>dbMergeSignal</u>	710
<u>Terminals</u>	711
<u>Member Terminals</u>	711
<u>Terminal Names</u>	711
<u>Terminal Connections</u>	711
<u>Terminal Direction</u>	712
<u>Terminal Position</u>	712
<u>Functions Acting on Terminals</u>	713
<u>dbCreateTerm</u>	713
<u>dbCreateBlockTerm</u>	713
<u>dbDeleteTerm</u>	713
<u>dbGetTermDirect</u>	714
<u>dbSetTermDirect</u>	714
<u>dbSetTermName</u>	714
<u>dbGetTermName</u>	714
<u>dbGetMemTermName</u>	715
<u>dbGetTermNet</u>	715
<u>dbGetTermNumBit</u>	715
<u>dbTermHasPin</u>	715
<u>dbStartGenTermToPin</u>	715
<u>dbGenTermToPin</u>	716
<u>dbGetTermRouteMethod</u>	717
<u>dbSetTermRouteMethod</u>	718
<u>dbGetTermPosition</u>	719
<u>dbSetTermPosition</u>	720
<u>dbUnsetTermPosition</u>	721
<u>dbGetTermsWithPosition</u>	722
<u>dbFindTermByPosition</u>	723
<u>Functions Acting on Terminals</u>	723

Cadence Integrators Toolkit Database Reference

<u>dbSetTermSoftConnect</u>	724
<u>dbIsTermSoftConnect</u>	724
<u>dbSetSoftConnectTermPinlessLayer</u>	725
<u>dbUnsetSoftConnectTermPinlessLayer</u>	725
<u>dbGetSoftConnectTermPinlessLayer</u>	725
<u>dbHasSoftConnectTermPinlessLayer</u>	725
<u>dbSetSoftConnectTermConnectToLayer</u>	726
<u>dbGetSoftConnectTermConnectToLayer</u>	726
<u>dbHasSoftConnectTermConnectToLayer</u>	726
<u>dbGetAnyTermGroundSensitivity</u>	727
<u>dbGetAnyTermSupplySensitivity</u>	728
<u>dbGetAnyTermGroundSensitivityAll</u>	729
<u>dbGetAnyTermSupplySensitivityAll</u>	730
<u>dbIsAnyTermGroundSensitivityConsistent</u>	731
<u>dbIsAnyTermSupplySensitivityConsistent</u>	732
<u>dbSetAnyTermGroundSensitivity</u>	733
<u>dbSetAnyTermSupplySensitivity</u>	734
<u>Terminal Antenna Functions</u>	735
<u>dbUUToDBUArea</u>	736
<u>dbUserToDBUArea</u>	736
<u>dbDBUAreaToUU</u>	736
<u>Terminal Antenna Rules</u>	736
<u>dbDeleteAllTerminalAntennaRules</u>	736
<u>dbHasTerminalAntennaRules</u>	737
<u>dbSetTerminalAntennaRules</u>	737
<u>dbDeleteTerminalAntennaRules</u>	737
<u>dbGetTerminalAntennaRules</u>	737
<u>dbStartGenTerminalAntennaRule</u>	738
<u>dbGenTerminalAntennaRule</u>	738
<u>dbSetTerminalAntennaRuleOnLayer</u>	738
<u>dbGetTerminalAntennaRuleOnLayer</u>	739
<u>dbDeleteTerminalAntennaRuleOnLayer</u>	739
<u>Instance Terminals</u>	740
<u>Connections for Instance Terminals</u>	740
<u>Functions Acting on Instance Terminals</u>	740
<u>dbCreateInstTerm</u>	741

Cadence Integrators Toolkit Database Reference

<u>dbCreateConn</u>	741
<u>dbCreateConnByName</u>	741
<u>dbCreateConnByNewName</u>	741
<u>dbDeleteInstTerm</u>	742
<u>dbDeleteConn</u>	742
<u>dbAddInstTermToNet</u>	742
<u>dbSubInstTermFromNet</u>	742
<u>dbIsInstTermBound</u>	743
<u>dbIsInstTermWidthOK</u>	743
<u>dbGetInstTermNet</u>	743
<u>dbGetInstTermAnyInst</u>	743
<u>dbGetInstTermTerm</u>	743
<u>dbGetInstTermNetName</u>	744
<u>dbGetInstTermInstName</u>	744
<u>dbGetInstTermTermName</u>	744
<u>dbGetMemInstTermMemInst</u>	745
<u>dbGetMemInstTermMemTerm</u>	745
<u>dbGetInstTermNumBit</u>	745
<u>dbCreateInstTermByPosition</u>	746
<u>dbGetInstTermTermPosition</u>	747
<u>Pins</u>	748
<u>dbCreatePin</u>	749
<u>dbCreatePinOnTerm</u>	750
<u>dbDeletePin</u>	750
<u>dbGetPinName</u>	750
<u>dbSetPinName</u>	751
<u>dbGetPinTerm</u>	751
<u>dbGetPinNet</u>	751
<u>dbMovePinToNet</u>	751
<u>dbGetPinFig</u>	752
<u>dbAddFigToPin</u>	752
<u>dbSubFigFromPin</u>	753
<u>dbStartGenPinToFig</u>	754

Cadence Integrators Toolkit Database Reference

<u>dbGenPinToFig</u>	755
<u>dbStartGenNetToTerm</u>	756
<u>dbGenNetToTerm</u>	757
<u>dbSetTermMustJoin</u>	758
<u>dbUnsetTermMustJoin</u>	759
<u>dbStartGenTermToMustJoinTerm</u>	760
<u>dbGenTermToMustJoinTerm</u>	761
<u>dbTermHasMustConnectAllPins</u>	762
<u>dbSetTermMustConnectAllPins</u>	763
<u>Access Directions of a Pin</u>	764
<u>dbGetPinAccessDirection</u>	764
<u>dbPinHasAccessDirection</u>	764
<u>dbSetPinAccessDirection</u>	765
<u>dbDeletePinAccessDirection</u>	765
<u>dbTransformAccessDirection</u>	765
<u>BusDef and BitOrder</u>	766
<u>dbCreateBusNetDef</u>	768
<u>dbCreateBusTermDef</u>	769
<u>dbCreateVectorInstDef</u>	770
<u>dbFindBusNetDef</u>	771
<u>dbFindBusTermDef</u>	772
<u>dbFindVectorInstDef</u>	773
<u>dbDeleteBusNetDef</u>	774
<u>dbDeleteBusTermDef</u>	775
<u>dbDeleteVectorInstDef</u>	776
<u>Attribute Access Functions</u>	777
<u>dbGetBusNetDefMinIndex</u>	777
<u>dbGetBusNetDefMaxIndex</u>	778
<u>dbGetBusNetDefBaseName</u>	779
<u>dbGetBusNetDefNumBits</u>	780
<u>dbGetBusNetDefNumBitsUsed</u>	781
<u>dbGetBusNetDefBitOrder</u>	782
<u>dbSetBusNetDefBitOrder</u>	783
<u>dbGetBusTermDefMinIndex</u>	784
<u>dbGetBusTermDefMaxIndex</u>	785
<u>dbGetBusTermDefBaseName</u>	786

Cadence Integrators Toolkit Database Reference

<u>dbGetBusTermDefNumBits</u>	787
<u>dbGetBusTermDefNumBitsUsed</u>	788
<u>dbGetBusTermDefBitOrder</u>	789
<u>dbSetBusTermDefBitOrder</u>	790
<u>dbGetVectorInstDefMinIndex</u>	791
<u>dbGetVectorInstDefMaxIndex</u>	792
<u>dbGetVectorInstDefBaseName</u>	793
<u>dbGetVectorInstDefNumBits</u>	794
<u>dbGetVectorInstDefNumBitsUsed</u>	795
<u>dbGetVectorInstDefBitOrder</u>	796
<u>dbSetVectorInstDefBitOrder</u>	797
<u>Cellview Information</u>	798
<u>dbCellViewHasOpticalData</u>	798
<u>dbCellViewHasOpticalData(cv_id)</u>	798
<u>dbDeleteAllNet</u>	798
<u>dbFindNetByName</u>	799
<u>dbFindSigByName</u>	799
<u>dbFindSigByAlias</u>	799
<u>dbFindAnyInstByName</u>	799
<u>dbFindMemInstByName</u>	799
<u>dbFindTermByName</u>	800
<u>dbHasTerm</u>	800
<u>dbStartGenTerm</u>	800
<u>dbGenTerm</u>	800
<u>dbStartGenMemTermByName</u>	800
<u>dbGenMemTermByName</u>	801
<u>dbStartGenNet</u>	801
<u>dbGenNet</u>	801
<u>dbStartGenMemInst</u>	801
<u>dbGenMemInst</u>	801
<u>dbStartGenSig</u>	802
<u>dbGenSig</u>	802
<u>dbStartGenConn</u>	802

Cadence Integrators Toolkit Database Reference

<u>dbGenConn</u>	802
<u>Equivalent Connectivity Time Functions</u>	803
<u>dbCellViewHasEquivalentConnectivityTime</u>	803
<u>dbGetCellViewEquivalentConnectivityTime</u>	804
<u>dbSetCellViewEquivalentConnectivityTime</u>	805
<u>dbUnsetCellViewEquivalentConnectivityTime</u>	806
<u>dbGetMemName</u>	806
<u>dbGetNameNumBit</u>	807
<u>dbProduceMemName</u>	807
<u>myMemNameConsumer</u>	807
<u>Net and Terminal Name Syntax</u>	808
<u>nameExpr</u>	808
<u>nameTerm</u>	808
<u>baseName</u>	808
<u>vectExpr</u>	808
<u>vectTerm</u>	809
<u>Name and Vector Expression Expansion</u>	810
<u>Automatic Database Deletions</u>	811
<u>Constraints on Database Objects</u>	811
<u>Restrictions to the Interface</u>	812
<u>Connectivity Example Function</u>	812

8

<u>CDBA Placement Functions</u>	815
<u>Structs in Placement Infrastructure</u>	816
<u>dbPlaceCellsAttrData</u>	822
<u>Enumerated Types and Enumerated Constants used in Placement Infrastructure</u>	824
<u>Background Definition Functions</u>	826
<u>dbAllocBackgroundDefAttrArray</u>	827
<u>dbCreateBackgroundDef</u>	828
<u>dbCreateBackgroundDefByAttr</u>	829
<u>dbDeleteBackgroundDef</u>	830
<u>dbFreeBackgroundDefAttrArray</u>	831
<u>dbGenBackgroundDef</u>	832
<u>dbGetBackgroundDefAttr</u>	833

Cadence Integrators Toolkit Database Reference

<u>dbGetBackgroundDefEnclosure</u>	834
<u>dbGetBackgroundDefLP</u>	835
<u>dbGetBackgroundDefName</u>	836
<u>dbGetBackgroundDefNetName</u>	837
<u>dbSetBackgroundDefEnclosures</u>	838
<u>dbSetBackgroundDefLP</u>	839
<u>dbSetBackgroundDefNetName</u>	840
<u>dbStartGenBackgroundDef</u>	841
<u>Component Type Set Definition Functions</u>	842
<u>dbAllocCompTypeSetDefAttrArray</u>	843
<u>dbCreateCompTypeSetDef</u>	844
<u>dbCreateCompTypeSetDefByAttr</u>	846
<u>dbFreeCompTypeSetDefAttrArray</u>	847
<u>dbGenCompTypeSetDef</u>	848
<u>dbGetCompTypeSetDefAttr</u>	849
<u>dbGetCompTypeSetDefCompFilters</u>	850
<u>dbSetCompTypeSetDefCompFilters</u>	851
<u>dbStartGenCompTypeSetDef</u>	852
<u>dbUnsetCompTypeSetDefCompFilters</u>	853
<u>Rail Definition Functions</u>	854
<u>dbAllocRailDefAttrArray</u>	855
<u>dbCreateRailDef</u>	856
<u>dbCreateRailDefByAttr</u>	858
<u>dbFreeRailDefAttrArray</u>	859
<u>dbGenRailDef</u>	860
<u>dbGetRailDefAttr</u>	861
<u>dbGetRailDefFigType</u>	862
<u>dbSetRailDefFigType</u>	863
<u>dbStartGenRailDef</u>	864
<u>Placement Area Definition Functions</u>	865
<u>dbAddPlaceAreaBackgroundDef</u>	866
<u>dbAddPlaceAreaCompTypeSetDef</u>	867
<u>dbAddPlaceAreaRailDef</u>	868

Cadence Integrators Toolkit Database Reference

<u>dbAllocPlaceRowAttrArray</u>	869
<u>dbCreatePlaceArea</u>	870
<u>dbFreePlaceRowAttrArray</u>	871
<u>dbGenPlaceArea</u>	872
<u>dbGenPlaceAreaToBackgroundDef</u>	873
<u>dbGenPlaceAreaToCompTypeSetDef</u>	874
<u>dbGenPlaceAreaToRailDef</u>	875
<u>dbGetPlaceAreaOrient</u>	876
<u>dbGetPlaceAreaUtilization</u>	877
<u>dbRemovePlaceAreaBackgroundDef</u>	878
<u>dbRemovePlaceAreaCompTypeSetDef</u>	879
<u>dbRemovePlaceAreaRailDef</u>	880
<u>dbSetPlaceAreaOrient</u>	881
<u>dbSetPlaceAreaUtilization</u>	882
<u>dbStartGenPlaceArea</u>	883
<u>dbStartGenPlaceAreaToBackgroundDef</u>	884
<u>dbStartGenPlaceAreaToCompTypeSetDef</u>	885
<u>dbStartGenPlaceAreaToRailDef</u>	886
<u>Row-based Functions</u>	887
<u>dbAddRowBackgroundDef</u>	889
<u>dbAddRowCompTypeSetDef</u>	890
<u>dbAddRowRailDef</u>	891
<u>dbCreateRow</u>	892
<u>dbDeleteRow</u>	893
<u>dbFindRowByName</u>	894
<u>dbGenRow</u>	895
<u>dbGenRowHeaderToRow</u>	896
<u>dbGenRowToBackgroundDef</u>	897
<u>dbGenRowToCompTypeSetDef</u>	898
<u>dbGenRowToRailDef</u>	899
<u>dbGetRow</u>	900
<u>dbStartGenRowToCompTypeSetDef</u>	901
<u>dbGetRowName</u>	902
<u>dbGetRowNumSites</u>	903
<u>dbGetRowSiteOrient</u>	904
<u>dbGetRowOrient</u>	905

Cadence Integrators Toolkit Database Reference

<u>dbGetRowOrigin</u>	906
<u>dbGetRowRowHeader</u>	907
<u>dbGetRowSiteDef</u>	908
<u>dbGetRowSiteDefName</u>	909
<u>dbGetRowSiteOrient</u>	910
<u>dbGetRowUtilization</u>	911
<u>dbIsRowBound</u>	912
<u>dbIsRowId</u>	913
<u>dbRemoveRowBackgroundDef</u>	914
<u>dbRemoveRowCompTypeSetDef</u>	915
<u>dbRemoveRowRailDef</u>	916
<u>dbSetRowName</u>	917
<u>dbSetRowNumSites</u>	918
<u>dbSetRowOrient</u>	919
<u>dbSetRowOrigin</u>	920
<u>dbSetRowSiteDef</u>	921
<u>dbSetRowSiteOrient</u>	922
<u>dbSetRowUtilization</u>	923
<u>dbStartGenRow</u>	924
<u>dbStartGenRowHeaderToRow</u>	925
<u>dbStartGenRowToBackgroundDef</u>	926
<u>dbStartGenRowToRailDef</u>	927
<u>Row Region Specification Functions</u>	928
<u>dbCreateRowRegionSpec</u>	929
<u>dbDeleteRowRegionSpec</u>	931
<u>dbFindRowRegionSpec</u>	932
<u>dbGenRowRegionSpecs</u>	933
<u>dbGenRowRegionSpecToRowRegion</u>	934
<u>dbGetRowRegionSpecGrids</u>	935
<u>dbGetRowRegionSpecHeight</u>	936
<u>dbGetRowRegionSpecName</u>	937
<u>dbGetRowRegionSpecOffsets</u>	938
<u>dbGetRowRegionSpecRelSnapPattern</u>	939

Cadence Integrators Toolkit Database Reference

<u>dbGetRowRegionSpecRowAttrs</u>	940
<u>dbSetRowRegionSpecGrids</u>	941
<u>dbSetRowRegionSpecHeight</u>	942
<u>dbSetRowRegionSpecOffset</u>	943
<u>dbSetRowRegionSpecRelSnapPattern</u>	944
<u>dbSetRowRegionSpecRowAttrs</u>	945
<u>dbStartGenRowRegionSpecs</u>	946
<u>dbStartGenRowRegionSpecToRowRegion</u>	947
<u>dbGetRowRegionSpecPlaceCells</u>	948
<u>dbSetRowRegionSpecPlaceCells</u>	949
<u>dbFreePlaceCellsListAttr</u>	950
Row Region Functions	951
<u>dbAttachRowRegionToPRBoundary</u>	952
<u>dbCreateRowRegion</u>	953
<u>dbRowRegionPointsCutOut</u>	954
<u>dbDetachPRBoundaryFromRowRegion</u>	955
<u>dbFindRowRegion</u>	956
<u>dbFlattenRowRegion</u>	957
<u>dbGenRowRegionPlaceRows</u>	958
<u>dbGenRowRegions</u>	959
<u>dbGetRowRegionAttachedToPRBoundary</u>	960
<u>dbGetRowRegionPoints</u>	961
<u>dbGetRowRegionRowRegionSpec</u>	962
<u>dbGetRowRegionUsesPartialRowSpec</u>	963
<u>dbIsRowRegion</u>	964
<u>dbIsRowRegionAttachedToPRBoundary</u>	965
<u>dbRebuildRowRegion</u>	966
<u>dbRowRegionMatchesSpec</u>	967
<u>dbSetRowRegionPoints</u>	968
<u>dbSetRowRegionRowRegionSpec</u>	969
<u>dbSetRowRegionUsesPartialRowSpec</u>	970
<u>dbStartGenRowRegionPlaceRows</u>	971
<u>dbStartGenRowRegions</u>	972
Miscellaneous Functions	973
<u>dbGetPlaceRowSoftwareVersion</u>	974
<u>dbGetPlaceRowDataVersion</u>	975

<u>dbGetRowRegionDataVersion</u>	976
<u>dbGetRowRegionSoftwareVersion</u>	977
<u>dbIsImplicit</u>	978
<u>dbIsInRowRegion</u>	979

9

[CDBA Parent-Child Relationship Functions](#)

<u>Limitations</u>	982
<u>Retrieval and Traversal Operations</u>	982
<u>dbStartGenChild</u>	982
<u>dbGenChild</u>	983
<u>dbGetParent</u>	983
<u>Predicates</u>	983
<u>dbIsParent</u>	983
<u>dbIsChild</u>	983
<u>dbIsChildOfParent</u>	984
<u>dbIsParentOfChild</u>	984
<u>dbHasMatchPointLists</u>	984
<u>Creation and Deletion Operations</u>	984
<u>dbAddChildToParent</u>	984
<u>dbSubChildFromParent</u>	985
<u>dbMatchPointLists</u>	985
<u>dbUnmatchPointLists</u>	985
<u>Other CDBA Functions Affected</u>	986

10

[CDBA Inherited Connection Functions](#)

<u>Signals and Inherited Net Expressions</u>	990
<u>dbCreateSigNetExpr</u>	991
<u>dbCVHasInheritedSig</u>	992
<u>dbDeleteCVInheritedSigs</u>	993
<u>dbDeleteSigNetExpr</u>	994

Cadence Integrators Toolkit Database Reference

<u>dbGenInheritedSig</u>	994
<u>dbGetSigNetExpr</u>	996
<u>dbIsSigInherited</u>	997
<u>dbIsValidSigNetExpr</u>	998
<u>dbParseSigNetExpr</u>	999
<u>dbReplaceSigNetExpr</u>	1001
<u>dbStartGenInheritedSig</u>	1002
<u>Terminals and Inherited Net Expressions</u>	1003
<u>dbCreateTermNetExpr</u>	1004
<u>dbCVHasInherited Term</u>	1005
<u>dbDeleteCVInheritedTerms</u>	1006
<u>dbDeleteTermNetExpr</u>	1006
<u>dbGenInheritedTerm</u>	1008
<u>dbGetTermNetExpr</u>	1009
<u>dbIsTermInherited</u>	1010
<u>dbIsValidTermNetExpr</u>	1011
<u>dbParseTermNetExpr</u>	1012
<u>dbReplaceTermNetExpr</u>	1014
<u>dbStartGenInheritedTerm</u>	1015

11

<u>CDF Read and Write Access Functions</u>	1017
<u>CDF Read-Access Functions</u>	1018
<u>Query Functions</u>	1019
<u>cdfGetLibCDF</u>	1019
<u>cdfGetBaseLibCDF</u>	1020
<u>cdfGetCellCDF</u>	1021
<u>cdfGetBaseCellCDF</u>	1022
<u>cdfGetInstCDF</u>	1023
<u>cdfGetCDFType</u>	1024
<u>CDF Data Retrieval Functions</u>	1025
<u>cdfFindParamByName</u>	1025
<u>cdfGetParamRec</u>	1026
<u>cdfGetDataRec</u>	1027
<u>Generator Functions</u>	1028

Cadence Integrators Toolkit Database Reference

<u>cdfStartGenParam</u>	1028
<u>cdfGenParam</u>	1029
<u>cdfStopGen</u>	1030
<u>CDF Write-Access Functions</u>	1031
<u>Functions to Create Data</u>	1031
<u>cdfCreateBaseCellCDF</u>	1031
<u>cdfCreateBaseLibCDF</u>	1034
<u>cdfCreateParam</u>	1037
<u>Functions to Delete Data</u>	1042
<u>cdfDeleteCDF</u>	1042
<u>cdfDeleteParam</u>	1043
<u>Functions to Copy Data</u>	1044
<u>cdfCopyCDF</u>	1044
<u>cdfCopyParam</u>	1045
<u>Functions to Update Data</u>	1046
<u>cdfUpdateInstParam</u>	1046
<u>cdfUpdateInstSingleParam</u>	1047
<u>cdfSyncInstParamValue</u>	1048
<u>The Function to Save Data</u>	1049
<u>cdfSaveCDF</u>	1049

12

<u>CDBA Relative Object Design Functions</u>	1051
<u>Overview</u>	1052
<u>About ROD Objects and ROD Object IDs</u>	1052
<u>Named Objects</u>	1053
<u>Hierarchical Name</u>	1053
<u>Boolean Query Functions</u>	1054
<u>rodIsMPPMaster</u>	1054
<u>rodIsMPRMaster</u>	1055
<u>rodIsMPPMasterChoppable</u>	1056
<u>rodIsMPPSubShape</u>	1057

Cadence Integrators Toolkit Database Reference

<u>rodIsObj</u>	1058
<u>rodIsObjAligned</u>	1059
<u>rodIsObjShape</u>	1060
<u>Object Query Functions</u>	1061
<u>rodGetObjByDbld</u>	1061
<u>rodGetObjByName</u>	1062
<u>rodGetObjDbld</u>	1063
<u>rodGetObjName</u>	1064
<u>rodGetObjTransform</u>	1065
<u>Multipart Path Functions</u>	1066
<u>rodDeleteAllMPPs</u>	1066
<u>rodExplodeMPP</u>	1067
<u>rodGetMPPSubShapeMaster</u>	1068
<u>Multipart Path Subshape Generation Functions</u>	1069
<u>rodStartGenMPPDblds</u>	1069
<u>rodGenMPPDbld</u>	1070
<u>rodStopGen</u>	1071
<u>Multipart Rectangle Functions</u>	1072
<u>rodGetMPRSubShapeMaster</u>	1072
<u>rodExplodeMPR</u>	1073
<u>rodReCreateMPR</u>	1074
<u>Multipart Rectangle Subshape Generation Functions</u>	1075
<u>rodGetMPRReclId</u>	1075
<u>rodStartGenMPRSubParts</u>	1076
<u>rodStartGenModMPRSubParts</u>	1077
<u>rodGenMPRSubParts</u>	1078
<u>rodGenMPRDbld</u>	1079
<u>rodStartGenMPRDblds</u>	1080
<u>rodStopGenMPR</u>	1081
<u>rodIsMPRSubShape</u>	1082
<u>Name, Unalign, Unname Functions</u>	1083
<u>rodNameShape</u>	1083
<u>rodUnAlign</u>	1085
<u>rodUnNameShape</u>	1086
<u>Handle Functions</u>	1087
<u>rodGetHandle</u>	1087

<u>rodGenStatId</u>	1089
<u>rodGenNamedShape</u>	1090

13

CDBA User Database Extensions

<u>Properties</u>	1093
<u>Property Types</u>	1093
<u>Property Creation and Modification Functions</u>	1095
<u>dbCreateProp</u>	1095
<u>dbCreateHierProp</u>	1095
<u>dbCreateRangeProp</u>	1096
<u>dbCreateEnumProp</u>	1097
<u>dbCopyProp</u>	1097
<u>dbCopySingleProp</u>	1098
<u>dbMergeProp</u>	1098
<u>dbMergeSingleProp</u>	1098
<u>dbResetPropList</u>	1099
<u>Property Information Replacement Functions</u>	1099
<u>dbReplaceProp</u>	1100
<u>dbReplacePropList</u>	1100
<u>dbReplaceInstParamList</u>	1100
<u>dbReplaceHierProp</u>	1101
<u>dbReplaceRangeProp</u>	1101
<u>dbReplaceEnumProp</u>	1101
<u>Property Information Retrieval Functions</u>	1102
<u>dbGetProp</u>	1102
<u>dbGetPropByName</u>	1103
<u>dbGetPropRange</u>	1104
<u>dbGetPropEnum</u>	1104
<u>Predicate Functions for Properties</u>	1106
<u>dbObjHasProp</u>	1106
<u>dbIsScalarProp</u>	1106

Cadence Integrators Toolkit Database Reference

<u>dbIsHierProp</u>	1106
<u>dbIsRangeProp</u>	1106
<u>dbIsEnumProp</u>	1107
<u>Property Deletion Functions</u>	1107
<u>dbDeleteProp</u>	1107
<u>dbDeletePropByName</u>	1107
<u>Property Generator Functions</u>	1108
<u>dbStartGenProp</u>	1108
<u>dbGenProp</u>	1108
<u>Groups</u>	1110
<u>Group Attributes</u>	1110
<u>Groups as Sets or Collections</u>	1110
<u>Deleting Groups</u>	1111
<u>Group Member Order</u>	1111
<u>Group Names</u>	1111
<u>Group Members</u>	1112
<u>Group Creation Functions</u>	1112
<u>dbCreateGroup</u>	1112
<u>dbCreateGroupWithDef</u>	1112
<u>Group Modification Functions</u>	1113
<u>dbAddIdToGroup</u>	1113
<u>dbDeleteGroupMemFromGroup</u>	1113
<u>dbDeletIdFromGroup</u>	1114
<u>dbDeletIdFromAllGroup</u>	1114
<u>dbMoveGroupMem</u>	1114
<u>dbSetGroupType</u>	1115
<u>dbSetGroupOrdered</u>	1115
<u>dbSetGroupUniqueName</u>	1115
<u>dbSetGroupDeleteWhen</u>	1115
<u>Group Information Retrieval Functions</u>	1116
<u>dbGetGroupName</u>	1116
<u>dbGetGroupGroupDefName</u>	1116
<u>dbGetGroupByName</u>	1116
<u>dbGetGroupType</u>	1116
<u>dbGetGroupOrdered</u>	1117
<u>dbGetGroupUniqueName</u>	1117

Cadence Integrators Toolkit Database Reference

<u>dbGetGroupDeleteWhen</u>	1117
<u>dbGetGroupMemGroup</u>	1117
<u>dbGetGroupMemId</u>	1117
<u>Predicate Functions for Groups</u>	1118
<u>Group Deletion Functions</u>	1118
<u>dbDeleteGroup</u>	1118
<u>dbDeleteGroupByName</u>	1118
<u>Group ID Generator Functions</u>	1120
<u>dbStartGenGroup</u>	1120
<u>dbStartGenGroupWithDef</u>	1120
<u>dbGenGroup</u>	1120
<u>dbGenGroupWithDef</u>	1121
<u>dbStartGenGroupByName</u>	1121
<u>dbGenGroupByName</u>	1121
<u>dbStartGenGroupTold</u>	1121
<u>dbGenGroupTold</u>	1122
<u>dbStartGenGroupToGroupMem</u>	1122
<u>dbGenGroupToGroupMem</u>	1122
<u>dbStartGenIdToGroup</u>	1122
<u>dbGenIdToGroup</u>	1123
<u>dbStartGenIdToGroupMem</u>	1123
<u>dbGenIdToGroupMem</u>	1123
<u>Pin Group Guides</u>	1124
<u>Functions for Adding, Removing, and Accessing Pins</u>	1124
<u>dbAddPinToPinGroupGuide</u>	1124
<u>dbSubPinFromPinGroupGuide</u>	1124
<u>dbAddFigToPinGroupGuide</u>	1124
<u>dbSubFigFromPinGroupGuide</u>	1125
<u>dbStartGenPinGroupGuidePins</u>	1125
<u>dbGenPinGroupGuideFigs</u>	1125
<u>dbDeletePinGroupGuide</u>	1125
<u>Functions for Accessing Pin Group Guide Attributes</u>	1126
<u>dbFindPinGroupGuideByName</u>	1126

Cadence Integrators Toolkit Database Reference

<u>dbGetPinGroupGuideType</u>	1126
<u>dbGetPinGroupGuideName</u>	1126
<u>dbSetPinGroupGuideName</u>	1126
<u>dbGetPinGroupGuidePreservePinOrder</u>	1127
<u>dbSetPinGroupGuidePreservePinOrder</u>	1127
<u>dbGetPinGroupGuideEvenPinSpread</u>	1127
<u>dbSetPinGroupGuideEvenPinSpread</u>	1127
<u>Functions for Specifying Pin Group Guide Constraints</u>	1128
<u>dbPinGroupGuideHasValidRoutingLayers</u>	1128
<u>dbGetPinGroupGuideValidRoutingLayers</u>	1128
<u>dbSetPinGroupGuideValidRoutingLayers</u>	1128
<u>dbUnsetPinGroupGuideValidRoutingLayers</u>	1128
<u>dbPinGroupGuideHasLayerPriorityOrder</u>	1129
<u>dbGetPinGroupGuideLayerPriorityOrder</u>	1129
<u>dbSetPinGroupGuideLayerPriorityOrder</u>	1129
<u>dbUnsetPinGroupGuideLayerPriorityOrder</u>	1129
<u>dbPinGroupGuideHasKeepoutSpacing</u>	1130
<u>dbGetPinGroupGuideKeepoutSpacing</u>	1130
<u>dbSetPinGroupGuideKeepoutSpacing</u>	1130
<u>dbUnsetPinGroupGuideKeepoutSpacing</u>	1130
<u>dbPinGroupGuideHasMinPinSpacing</u>	1131
<u>dbGetPinGroupGuideMinPinSpacing</u>	1131
<u>dbSetPinGroupGuideMinPinSpacing</u>	1131
<u>dbUnsetPinGroupGuideMinPinSpacing</u>	1131
<u>dbPinGroupGuideHasMinPinWidth</u>	1132
<u>dbGetPinGroupGuideMinPinWidth</u>	1132
<u>dbSetPinGroupGuideMinPinWidth</u>	1132
<u>dbUnsetPinGroupGuideMinPinWidth</u>	1132
<u>Area Pin Group Guides Functions</u>	1133
<u>dbCreateAreaPinGroupGuide</u>	1133
<u>dbAddAreaToPinGroupGuide</u>	1133
<u>dbSubAreaFromPinGroupGuide</u>	1133
<u>dbGetAreaBoundaryPinGroupGuide</u>	1133
<u>dbAreaBoundaryHasValidRoutingLayers</u>	1134
<u>dbGetAreaBoundaryValidRoutingLayers</u>	1134
<u>dbSetAreaBoundaryValidRoutingLayers</u>	1134

Cadence Integrators Toolkit Database Reference

<u>dbUnsetAreaBoundaryValidRoutingLayers</u>	1134
<u>dbAreaBoundaryHasLayerPriorityOrder</u>	1135
<u>dbGetAreaBoundaryLayerPriorityOrder</u>	1135
<u>dbSetAreaBoundaryLayerPriorityOrder</u>	1135
<u>dbUnsetAreaBoundaryLayerPriorityOrder</u>	1135
<u>dbStartGenPinGroupGuideAreas</u>	1136
<u>dbGenPinGroupGuideAreas</u>	1136
Edge Pin Group Guide Functions	1136
<u>dbCreateEdgePinGroupGuide</u>	1136
<u>dbGetPinGroupGuideEdgeBoundary</u>	1136
<u>dbGetPinGroupGuideEdgeName</u>	1137
<u>dbGetPinGroupGuideEdgeStartOffset</u>	1137
<u>dbGetPinGroupGuideEdgeStopOffset</u>	1137
<u>dbSetPinGroupGuideEdgeStartOffset</u>	1137
<u>dbSetPinGroupGuideEdgeStopOffset</u>	1137
<u>dbRemovePinGroupGuideEdge</u>	1138
<u>dbSetPinGroupGuideEdge</u>	1138
Pin Group Guide Functions for Other Objects	1138
<u>dbPinHasMinPinSpacing</u>	1138
<u>dbGetPinMinPinSpacing</u>	1138
<u>dbSetPinMinPinSpacing</u>	1139
<u>dbUnsetPinMinPinSpacing</u>	1139
<u>dbPinHasMinPinWidth</u>	1139
<u>dbGetPinMinPinWidth</u>	1139
<u>dbSetPinMinPinWidth</u>	1140
<u>dbUnsetPinMinPinWidth</u>	1140
<u>dbGetFigPinGroupGuide</u>	1140
<u>dbStartGenCellViewPinGroupGuides</u>	1140
<u>dbGenCellViewPinGroupGuides</u>	1141
<u>dbStartGenPinToPinGroupGuides</u>	1141
<u>dbGenPinToPinGroupGuides</u>	1141
<u>dbStartGenPRBoundaryPinGroupGuides</u>	1141
<u>dbGenPRBoundaryPinGroupGuides</u>	1141

14

<u>CDBA Technology File Functions</u>	1143
<u>Technology File Overview</u>	1145
<u>Uses for Technology File Functions</u>	1145
<u>Accessing Technology Data</u>	1145
<u>Technology File Constraints</u>	1146
<u>Warnings in Functions</u>	1146
<u>Speed of Implementation in Functions</u>	1147
<u>Ordered Properties in Functions</u>	1147
<u>Pointers in Technology File Functions</u>	1148
<u>Attach Functions</u>	1148
<u>Administrative (File) Functions</u>	1149
<u>techCloseTechFile</u>	1150
<u>techCopyTechFile</u>	1151
<u>techDeleteTechFile</u>	1152
<u>techGetCreateTime</u>	1153
<u>techGetTechFileDdId</u>	1154
<u>techGetTimeStamp</u>	1155
<u>techOpenDefaultTechFile;</u>	1156
<u>techOpenTechFile</u>	1157
<u>techPurgeTechFile</u>	1159
<u>techRefreshTechFile</u>	1160
<u>techReopenTechFile</u>	1161
<u>techSaveTechFile</u>	1162
<u>techSaveTechFilesInITDBGGraph</u>	1163
<u>techTruncateTechFile</u>	1164
<u>techVerifyTechFileId</u>	1165
<u>Technology File Management Functions</u>	1166
<u>*FuncConflictTriggerPtr</u>	1167
<u>*FuncGraphTriggerPtr</u>	1169
<u>techDeleteCellViewTechLibName</u>	1170
<u>techDeleteObjTechLibName</u>	1171
<u>techHasConflict</u>	1172
<u>techExit</u>	1173
<u>techGetCellViewDeviceTechFile</u>	1174

Cadence Integrators Toolkit Database Reference

<u>techGetCellViewTechFile</u>	1175
<u>techGetCellViewTechLibName</u>	1176
<u>techGetDefTechLibName</u>	1177
<u>techGetObjTechFile</u>	1178
<u>techGetObjTechLibName</u>	1179
<u>techGetReadFileName</u>	1180
<u>techGetTechFileMode</u>	1181
<u>techGetTechFileName</u>	1182
<u>techGetTechLibName</u>	1183
<u>techGetWriteFileName</u>	1184
<u>techIdToList</u>	1185
<u>techInit</u>	1186
<u>techIsLPId</u>	1187
<u>techRegConflictTrigger</u>	1188
<u>techRegGraphModTrigger</u>	1189
<u>techSetCellViewTechLibName</u>	1190
<u>techSetObjTechLibName</u>	1191
<u>techStringToPurpose</u>	1192
<u>techUnattachTechFile</u>	1193
<u>techUnregConflictTrigger</u>	1194
<u>techUnregGraphModTrigger</u>	1195
<u>Control Class Functions</u>	1196
<u>techGenAllRef</u>	1197
<u>techGenAllRefLibName</u>	1198
<u>techGenParam</u>	1199
<u>techGenRef</u>	1200
<u>techGenRefLibName</u>	1201
<u>techGetDerivedLayerLayer1Num</u>	1202
<u>techGetDerivedLayerLayer2Num</u>	1203
<u>techGetLayerMfgResolution</u>	1204
<u>techGetParam</u>	1205
<u>techIsMfgGridResolutionSet</u>	1206
<u>techMakeEffectiveLPsLocal</u>	1207

Cadence Integrators Toolkit Database Reference

<u>techPurposeDefHasDescription</u>	1208
<u>techPurposeDefHasParent</u>	1209
<u>techPurposeDefHasSigType</u>	1210
<u>techPurposeDefHasVoltageRange</u>	1211
<u>techSetEvaluateFlag</u>	1212
<u>techStartGenAllRefLibName</u>	1213
<u>techSetLayerMfgResolution</u>	1214
<u>techSetParam</u>	1215
<u>techSetRefs</u>	1216
<u>techStartGenAllRef</u>	1217
<u>techStartGenParam</u>	1218
<u>techStartGenRef</u>	1219
<u>techStartGenRefLibName</u>	1220
<u>techStopRuleGen</u>	1221
<u>techUnsetMfgGridResolution</u>	1222
<u>techUnsetRefs</u>	1223
<u>Layer Functions</u>	1224
<u>Rules for Layers and Layer-Purpose Pairs</u>	1224
<u>Equivalent Layers</u>	1225
<u>Stream Layer Rules</u>	1225
<u>Function Performance</u>	1226
<u>Equivalent Layer Functions</u>	1227
<u>techStartGenEquivLayerRule</u>	1227
<u>techGenEquivLayerRule</u>	1228
<u>techStartGenEquivLayer</u>	1229
<u>techGenEquivLayer</u>	1230
<u>techSetEquivLayer</u>	1231
<u>Via Layers Functions</u>	1232
<u>techStartGenViaLayer</u>	1232
<u>techGenViaLayer</u>	1233
<u>techSetViaLayer</u>	1234
<u>techIsViaLayer</u>	1235
<u>Derived Layers</u>	1236
<u>techGetNumTechLayers</u>	1238
<u>techIsLayerId</u>	1239
<u>techGetTechLayerIdByNumber</u>	1240

Cadence Integrators Toolkit Database Reference

<u>techGetTechLayerNumber</u>	1241
<u>techGetTechLayerName</u>	1242
<u>techIsPhysicalLayerId</u>	1243
<u>techIsDerivedLayerId</u>	1244
<u>techCreateDerivedLayer</u>	1245
<u>techStartGenDerivedLayer</u>	1246
<u>techGenDerivedLayer</u>	1247
<u>techGetDerivedTechLayer</u>	1248
<u>techGetDerivedLayerNum</u>	1249
<u>techGetDerivedTechLayerSpec</u>	1250
<u>techGetDerivedLayerSpecByNumber</u>	1251
<u>techGetLayerNum</u>	1252
<u>techGetLayerNameByNumber</u>	1253
<u>techGetLayerProp</u>	1254
<u>techGetLayerPropByNumber</u>	1255
<u>techLayerIsReserved</u>	1256
<u>techGetLayerAbbrev</u>	1257
<u>techSetLayerAbbrev</u>	1258
<u>techSetLayerName</u>	1259
<u>techSetLayerMfgResolution</u>	1260
<u>techGetLayerMfgResolution</u>	1261
<u>techGetLayerMaskNumber</u>	1262
<u>techSetLayerMaskNumber</u>	1263
<u>techStartGenLayerRoutingGrid</u>	1264
<u>techGenLayerRoutingGrid</u>	1265
<u>techGetLayerRoutingDirection</u>	1267
<u>techSetLayerRoutingDirection</u>	1268
<u>techGetLayerRoutingPitch</u>	1269
<u>techSetLayerRoutingPitch</u>	1270
<u>techGetLayerRoutingOffset</u>	1271
<u>techSetLayerRoutingOffset</u>	1272
<u>techStartGenLayerFunction</u>	1273
<u>techGenLayerFunction</u>	1274

Cadence Integrators Toolkit Database Reference

<u>techGenLayerFunctionAndMaskNumber</u>	1275
<u>Layer-Purpose (LP) Pair Functions</u>	1276
<u>techContainsLP</u>	1277
<u>techCreateDerivedPurposeDef</u>	1278
<u>techCreateLP</u>	1279
<u>techCreatePurposeDef</u>	1281
<u>techCreateTwoLayerProp</u>	1282
<u>techCreateTwoLayerPropByNum</u>	1283
<u>techDeleteLP</u>	1284
<u>techDeletePurposeDef</u>	1285
<u>techDeleteTwoLayerProp</u>	1286
<u>techDeleteTwoLayerPropByNum</u>	1287
<u>techGenLP</u>	1288
<u>techGenLPId</u>	1289
<u>techGenPurposeDef</u>	1290
<u>techGetLayerProp</u>	1291
<u>techGetLayerPropByNumber</u>	1292
<u>techGetLP</u>	1293
<u>techGetLPChangeLayer</u>	1294
<u>techGetLPDragEnable</u>	1295
<u>techGetLPLayerName</u>	1296
<u>techGetLPLayerProp</u>	1297
<u>techGetLPName</u>	1298
<u>techGetLPNumById</u>	1299
<u>techGetLPPacketName</u>	1300
<u>techGetLPPurposeDef</u>	1301
<u>techGetLPPriority</u>	1302
<u>techGetLPPriorityInContext</u>	1303
<u>techGetLPProp</u>	1304
<u>techGetLPPurpose</u>	1305
<u>techGetLPSelectable</u>	1306
<u>techGetLPTechFile</u>	1307
<u>techGetLPValid</u>	1308
<u>techGetLPVisible</u>	1309
<u>techGetPurposeAbbrev</u>	1310
<u>techGetPurposeDef</u>	1311

Cadence Integrators Toolkit Database Reference

<u>techGetPurposeDefAbbrev</u>	1312
<u>techGetPurposeDefDescription</u>	1313
<u>techGetPurposeDefName</u>	1314
<u>techGetPurposeDefNumber</u>	1315
<u>techGetPurposeDefParent</u>	1316
<u>techGetPurposeDefSigType</u>	1317
<u>techGetPurposeDefVoltageRange</u>	1318
<u>techGetPurposeName</u>	1319
<u>techFindTwoLayerProp</u>	1320
<u>techFindTwoLayerPropByNum</u>	1322
<u>techIsReservedPurposeDef</u>	1323
<u>techList2Layer</u>	1324
<u>techListToLayer</u>	1325
<u>techList2LP</u>	1326
<u>techListToLP</u>	1328
<u>techList2LPNoWarn</u>	1330
<u>techList2TechFile</u>	1332
<u>techListToTechFile</u>	1333
<u>techMakeEffectiveLPsLocal</u>	1334
<u>techPurposeToString</u>	1335
<u>techReplaceTwoLayerProp</u>	1336
<u>techReplaceTwoLayerPropByNum</u>	1337
<u>techSetLayerPropByNumber</u>	1338
<u>techSetLayerProp</u>	1339
<u>techSetLPChangeLayer</u>	1340
<u>techSetLPDragEnable</u>	1341
<u>techSetLPLayerProp</u>	1342
<u>techSetLPPacketName</u>	1343
<u>techSetLPPriority</u>	1344
<u>techSetLPPriorityInContext</u>	1345
<u>techSetLPProp</u>	1346
<u>techSetLPSelectable</u>	1347
<u>techSetLPValid</u>	1348

Cadence Integrators Toolkit Database Reference

<u>techSetLPVisible</u>	1349
<u>techSetPurposeAbbrev</u>	1350
<u>techSetPurposeDefAbbrev</u>	1351
<u>techSetPurposeDefDescription</u>	1352
<u>techSetPurposeDefName</u>	1353
<u>techSetPurposeDefParent</u>	1354
<u>techSetPurposeDefSigType</u>	1355
<u>techSetPurposeDefVoltageRange</u>	1356
<u>techSetPurposeName</u>	1357
<u>techStartGenLPId</u>	1358
<u>techStartGenLP</u>	1359
<u>techStartGenPurposeDef</u>	1360
<u>techUnsetPurposeDefDescription</u>	1361
<u>techUnsetPurposeDefParent</u>	1362
<u>techUnsetPurposeDefSigType</u>	1363
<u>techUnsetPurposeDefVoltageRange</u>	1364
<u>techVerifyTechFileId</u>	1365
<u>Layout Suite L Rule Functions</u>	1366
<u>techStartGenLeLswLayer</u>	1366
<u>techGenLeLswLayer</u>	1367
<u>techSetLeLswLayer</u>	1368
<u>Virtuoso Layout Suite XL Functions</u>	1369
<u>techStartGenLxExtractLayer</u>	1369
<u>techGenLxExtractLayer</u>	1371
<u>techSetLxExtractLayer</u>	1372
<u>techIsLxExtractLayer</u>	1373
<u>techStartGenLxNoOverlapLayer</u>	1374
<u>techGenLxNoOverlapLayer</u>	1375
<u>techSetLxNoOverlapLayer</u>	1376
<u>techIsLxNoOverlapLayer</u>	1377
<u>Place and Route Rule Functions</u>	1378
<u>prRoutingLayers</u>	1378
<u>techStartGenPrRoutingLayer</u>	1378
<u>techGenPrRoutingLayer</u>	1379
<u>techSetPrRoutingLayer</u>	1380
<u>techGetPrRoutingDirection</u>	1382

Cadence Integrators Toolkit Database Reference

<u>techIsPrRoutingLayer</u>	1383
<u>techGetPrRoutingPitch</u>	1384
<u>techSetPrRoutingPitch</u>	1386
<u>techGetPrRoutingOffset</u>	1387
<u>techSetPrRoutingOffset</u>	1389
<u>prViaTypes</u>	1390
<u>techStartGenPrViaType</u>	1390
<u>techGenPrViaType</u>	1391
<u>techSetPrViaType</u>	1392
<u>techGetPrViaType</u>	1394
<u>techIsPrViaDevice</u>	1396
<u>prOverlapLayers</u>	1397
<u>techStartGenPrOverlapLayer</u>	1397
<u>techGenPrOverlapLayer</u>	1398
<u>techIsPrOverlapLayer</u>	1399
<u>techSetPrOverlapLayer</u>	1400
<u>prViaStacks</u>	1401
<u>techStartGenPrStackVia</u>	1401
<u>techGenPrStackVia</u>	1402
<u>techSetPrStackVia</u>	1403
<u>techIsPrStackVia</u>	1404
<u>prViaMaxStackVias</u>	1405
<u>techStartGenPrMaxStackVia</u>	1405
<u>techGenPrMaxStackVia</u>	1406
<u>techSetPrMaxStackVia</u>	1407
<u>prMastersliceLayers</u>	1408
<u>techStartGenPrMastersliceLayer</u>	1408
<u>techGenPrMastersliceLayer</u>	1409
<u>techSetPrMastersliceLayer</u>	1410
<u>techIsPrMastersliceLayer</u>	1411
<u>prViaRules</u>	1412
<u>techStartGenPrViaRule</u>	1412
<u>techGenPrViaRule</u>	1412

Cadence Integrators Toolkit Database Reference

<u>techSetPrViaRule</u>	1413
<u>prGenViaRules</u>	1415
<u>techStartGenPrGenViaRule</u>	1415
<u>techSetPrGenViaRule</u>	1415
<u>techGen</u>	1416
<u>techStartGenPrTurnViaRule</u>	1417
<u>techGenPrTurnViaRule</u>	1418
<u>prNonDefaultRules</u>	1419
<u>techStartGenPrNonDefaultRule</u>	1419
<u>techGenPrNonDefaultRule</u>	1419
<u>techSetPrNonDefaultRule</u>	1419
<u>techStartGenPrNonDefaultParam</u>	1420
<u>techGenPrNonDefaultParam</u>	1421
<u>viaDef Functions</u>	1423
<u>About viaDefs</u>	1423
<u>techList2ViaDef</u>	1424
<u>techIsViaDefId</u>	1425
<u>techIsCustomViaDefId</u>	1426
<u>techIsStandardViaDefId</u>	1427
<u>techFindViaDefByName</u>	1428
<u>techDeleteViaDef</u>	1429
<u>techGetViaDefName</u>	1430
<u>techCreateCustomViaDef</u>	1431
<u>techCreateCustomViaDefByName</u>	1432
<u>techGetCustomViaDefLibName</u>	1433
<u>techGetCustomViaDefCellName</u>	1434
<u>techGetCustomViaDefViewName</u>	1435
<u>techGetViaDefLayers</u>	1436
<u>techGetViaDefLayer1</u>	1437
<u>techGetViaDefLayer1Num</u>	1438
<u>techGetViaDefLayer2Num</u>	1439
<u>techGetViaDefLayer2</u>	1440
<u>techViaDefHasLayer</u>	1441
<u>techGetViaDefResistancePerCut</u>	1442
<u>techSetViaDefResistancePerCut</u>	1443
<u>techGetNumViaDefs</u>	1444

Cadence Integrators Toolkit Database Reference

<u>techStartGenViaDef</u>	1445
<u>techGetStdViaDefImplant2Num</u>	1446
<u>techGenViaDef</u>	1447
<u>techCreateStdViaDef</u>	1448
<u>techStdViaDefHasImplant1</u>	1450
<u>techStdViaDefHasImplant2</u>	1451
<u>techGetStdViaDefImplant1</u>	1452
<u>techGetStdViaDefImplant2</u>	1453
<u>techGetStdViaDefParams</u>	1454
<u>techSetStdViaDefParams</u>	1455
<u>viaSpec Functions</u>	1456
<u>About Via Specifications</u>	1456
<u>techGetViaSpecDefaultViaDefNames</u>	1457
<u>viaHeader Functions</u>	1458
<u>About viaHeaders</u>	1458
<u>dbIsViaHeaderId</u>	1459
<u>dbFindViaHeaderByName</u>	1460
<u>dbFindViaHeader</u>	1461
<u>dbIsViaHeaderBound</u>	1462
<u>dbGetViaHeaderViaDef</u>	1463
<u>dbGetViaHeaderViaDefName</u>	1464
<u>dbStartGenViaHeader</u>	1465
<u>dbGenViaHeader</u>	1466
<u>dbGetViaHeaderMaster</u>	1467
<u>dbGetViaHeaderMasterBBox</u>	1468
<u>dbIsViaSuperHeader</u>	1469
<u>dbIsViaSubHeader</u>	1470
<u>dbStartGenViaSuperHeaderToSubHeader</u>	1471
<u>dbGenViaSuperHeaderToSubHeader</u>	1472
<u>dbGetStdViaHeaderParams</u>	1473
<u>dbGetCustomViaHeaderParams</u>	1474
<u>dbGetCustomViaHeaderAllParams</u>	1475
<u>dbGetCustomViaHeaderLibName</u>	1476

Cadence Integrators Toolkit Database Reference

<u>dbGetCustomViaHeaderCellName</u>	1477
<u>dbGetCustomViaHeaderViewName</u>	1478
<u>Site Definition Functions</u>	1479
<u>About siteDefs</u>	1479
<u>techIsSiteDefId</u>	1481
<u>techDeleteSiteDef</u>	1482
<u>techFindSiteDefByName</u>	1483
<u>techGetSiteDefName</u>	1484
<u>techGetSiteDefType</u>	1485
<u>techGetSiteDefWidth</u>	1486
<u>techGetSiteDefHeight</u>	1487
<u>techIsSiteDefSymmetricInX</u>	1488
<u>techIsSiteDefSymmetricInY</u>	1489
<u>techIsSiteDefSymmetricInR90</u>	1490
<u>techSetSiteDefSymmetricInX</u>	1491
<u>techSetSiteDefSymmetricInY</u>	1492
<u>techSetSiteDefSymmetricInR90</u>	1493
<u>techSetSiteDefType</u>	1494
<u>techStartGenSiteDef</u>	1495
<u>techGenSiteDef</u>	1496
<u>techGenSiteDefToRowHeader</u>	1497
<u>techStartGenCellViewSiteDefs</u>	1498
<u>techGenCellViewSiteDefs</u>	1499
<u>techStringToSiteDefType</u>	1500
<u>techSiteDefTypeToString</u>	1501
<u>techListToSiteDefType</u>	1502
<u>techList2SiteDef</u>	1503
<u>techIsScalarSiteDefId</u>	1504
<u>techCreateScalarSiteDef</u>	1505
<u>techSetScalarSiteDefWidth</u>	1506
<u>techSetScalarSiteDefHeight</u>	1507
<u>techIsArraySiteDefId</u>	1508
<u>techCreateArraySiteDef</u>	1509
<u>Site Pattern Functions</u>	1511
<u>techGetArraySiteDefSitePattern</u>	1512
<u>techSetArraySiteDefSitePattern</u>	1513

Cadence Integrators Toolkit Database Reference

<u>techCopySitePattern</u>	1514
<u>techFreeSitePattern</u>	1515
<u>dbGetCellViewSitePattern</u>	1516
<u>dbSetCellViewSitePattern</u>	1517
<u>Physical Constraint Functions</u>	1518
<u>techStartGenSpacingRule</u>	1519
<u>techGenSpacingRule</u>	1520
<u>techGetSpacingRule</u>	1522
<u>techSetSpacingRule</u>	1524
<u>techStartGenOrderedSpacingRule</u>	1526
<u>techGenOrderedSpacingRule</u>	1527
<u>techGetOrderedSpacingRule</u>	1528
<u>techSetOrderedSpacingRule</u>	1529
<u>Electrical Rule Functions</u>	1530
<u>techStartGenElectricalRule</u>	1531
<u>techGenElectricalRule</u>	1532
<u>techGetElectricalRule</u>	1533
<u>techSetElectricalRule</u>	1535
<u>techStartGenOrderedElectricalRule</u>	1537
<u>techGenOrderedElectricalRule</u>	1538
<u>techGetOrderedElectricalRule</u>	1539
<u>techSetOrderedElectricalRule</u>	1541
<u>Technology File Database Functions</u>	1543
<u>techGetDBUPerUU</u>	1543
<u>techGetUserUnit</u>	1544
<u>techIsDBUPerUUSet</u>	1545
<u>techIsUserUnitSet</u>	1546
<u>techSetDBUPerUU</u>	1547
<u>techSetUserUnit</u>	1548
<u>techUnsetDBUPerUU</u>	1549
<u>techUnsetUserUnit</u>	1550
<u>Trim Layer Functions</u>	1551
<u>techGetTrimLayer</u>	1553

Cadence Integrators Toolkit Database Reference

<u>techGetTrimmedLayers</u>	1555
<u>techFreeColoredLayerArray</u>	1557
<u>dbGetShapeTrimFillType</u>	1558
<u>dbStartGenCellViewTrimFillShapes</u>	1559
<u>dbGenCellViewTrimFillShapes</u>	1560

15

CDBA Display Resource File Functions 1561

Moving the Display Resource Definition 1562

Using DRM Software 1562

Displays 1563

drCreateDisplay 1563

drDeleteDisplay 1564

drGetDisplayId 1565

drIsDisplayId 1566

drGetDisplayName 1567

drGetDisplayMaxColors 1568

drGetDisplayMaxStipples 1569

drGetDisplayMaxLineStyle 1570

drStartGenDisplay 1571

drGenDisplay 1572

Display-Dependent Attributes 1573

drGetPacketFillColor 1574

drGetPacketFillColorIndex 1575

drGetPacketFillColorName 1576

drSetPacketFillColor 1577

drGetPacketOutlineColor 1578

drGetPacketOutlineColorIndex 1579

drGetPacketOutlineColorName 1580

drSetPacketOutlineColor 1581

drGetPacketStipple 1582

drGetPacketStippleIndex 1583

drGetPacketStippleName 1584

drGetPacketFillStyle 1585

drSetPacketFillStyle 1586

Cadence Integrators Toolkit Database Reference

<u>drSetPacketStipple</u>	1587
<u>drGetPacketLineStyle</u>	1588
<u>drGetPacketLineStyleIndex</u>	1589
<u>drGetPacketLineStyleName</u>	1590
<u>drSetPacketLineStyle</u>	1591
<u>drSetStipple</u>	1592
<u>drFreeStipple</u>	1593
<u>drSetLineStyle</u>	1594
<u>drFreeLineStyle</u>	1595
<u>drSetPacket</u>	1596
<u>drmInit</u>	1598
<u>drOpenDrf</u>	1599
<u>drSave</u>	1600
<u>Colors</u>	1601
<u>General Data Structures to Generate Colors</u>	1601
<u>drColorStructure</u>	1602
<u>drMakeColor</u>	1603
<u>drGetNumberOfColors</u>	1604
<u>drSetColorBlink</u>	1605
<u>drGetColorBlink</u>	1606
<u>drGetColor</u>	1607
<u>drGetColorByName</u>	1608
<u>drGetColorIndexByName</u>	1609
<u>drStartGenColor</u>	1610
<u>drGenColor</u>	1611
<u>Functions to Generate DRM Colors</u>	1612
<u>Stipple Patterns</u>	1613
<u>drStipple</u>	1614
<u>drMakeStipple</u>	1615
<u>drGetStipple</u>	1616
<u>drGetStippleByName</u>	1617
<u>drDeleteStippleByName</u>	1618
<u>drGetStippleIndexByName</u>	1619

Cadence Integrators Toolkit Database Reference

<u>drStartGenStipple</u>	1620
<u>Description</u>	1621
<u>drGetNumberOfStipple</u>	1622
<u>drGenStipple</u>	1623
<u>Line Styles</u>	1624
<u>drLineStyle</u>	1625
<u>drMakeLineStyle</u>	1626
<u>drGetLineStyle</u>	1627
<u>drGetLineStyleByName</u>	1628
<u>drGetLineStyleIndexByName</u>	1629
<u>drStartGenLineStyle</u>	1630
<u>drGenLineStyle</u>	1631
<u>Packets</u>	1632
<u>drMakePacket</u>	1633
<u>drGetPacket</u>	1634
<u>drDeletePacket</u>	1635
<u>drMakePacketAlias</u>	1636
<u>drGetNumberOfPackets</u>	1637
<u>drGetPacketIndex</u>	1638
<u>drFindPacket</u>	1639
<u>drMakeDefaultPacket</u>	1640
<u>drStartGenPacketIndex</u>	1641
<u>drGenPacket</u>	1642

16

<u>CDBA Hierarchy Database Functions</u>	1643
<u>HDB Functions</u>	1643
<u>Related Information</u>	1646

17

<u>CST Access C Functions</u>	1649
<u>Constraint Functions</u>	1650
<u>cstCreateLayerConstraint</u>	1651
<u>cstCreateLayerPairConstraint</u>	1653
<u>cstCreateLayerArrayConstraint</u>	1655

Cadence Integrators Toolkit Database Reference

<u>cstCreateSimpleConstraint</u>	1657
<u>cstAddToConstraintGroup</u>	1659
<u>cstReplaceIdsInConstraintGroup</u>	1660
<u>cstDeleteConstraint</u>	1661
<u>cstGetType</u>	1662
<u>cstGetConstraintDatabase</u>	1663
<u>cstGetConstraintDefName</u>	1664
<u>cstGetConstraintName</u>	1665
<u>cstGetConstraintDescription</u>	1666
<u>cstGetConstraintID</u>	1667
<u>cstGetLayerConstraintLayer</u>	1668
<u>cstGetLayerPairConstraintLayers</u>	1669
<u>cstGetLayerArrayConstraintLayers</u>	1670
<u>cstGetConstraintValue</u>	1671
<u>cstSetConstraintValue</u>	1672
<u>cstGetConstraintParams</u>	1673
<u>cstSetConstraintParams</u>	1674
<u>cstIsHardConstraint</u>	1675
<u>cstSetHardConstraint</u>	1676
<u>cstSetConstraintID</u>	1677
<u>cstSetConstraintDescription</u>	1678
<u>Constraint Group Functions</u>	1679
<u>cstCreateConstraintGroupOn</u>	1679
<u>cstFindConstraintGroupOn</u>	1681
<u>cstCreateConstraintGroupInTech</u>	1682
<u>cstCreateConstraintGroupInCellview</u>	1683
<u>cstFindConstraintGroupInTech</u>	1684
<u>cstFindConstraintGroupInCellview</u>	1685
<u>cstGetConstraintGroupOwner</u>	1686
<u>cstGetConstraintGroupDatabase</u>	1687
<u>cstDeleteConstraintGroup</u>	1688
<u>cstAccessGetConstraintGroupName</u>	1689
<u>cstGetConstraintGroupDefName</u>	1690

Cadence Integrators Toolkit Database Reference

<u>cstGetConstraintGroupOperator</u>	1691
<u>Generator Functions</u>	1692
<u>cstStartGenConstraintGroupTold</u>	1693
<u>cstGenConstraintGroupTold</u>	1694
<u>cstStartGenTechConstraintGroup</u>	1695
<u>cstGenTechConstraintGroup</u>	1696
<u>cstStartGenCellViewConstraintGroup</u>	1697
<u>cstGenCellViewConstraintGroup</u>	1698
<u>cstStartGenConstraintConstraintGroup</u>	1699
<u>cstGenConstraintConstraintGroup</u>	1700
<u>cstStartGenTechUnreferencedConstraint</u>	1701
<u>cstGenTechUnreferencedConstraint</u>	1702
<u>cstStartGenCellViewUnreferencedConstraint</u>	1703
<u>cstGenCellViewUnreferencedConstraint</u>	1704
<u>cstStopGen</u>	1705
<u>20nm CST APIs</u>	1706
<u>cstFindFirstLayerConstraint</u>	1707
<u>cstFindFirstLayerPairConstraint</u>	1709
<u>cstFindFirstLayerArrayConstraint</u>	1711
<u>cstFindFirstSimpleConstraint</u>	1713
<u>cstGet1DTableValue</u>	1714
<u>cstGet2DTableValue</u>	1715
<u>cstGetTwoWidthTableValue</u>	1716

18

InPlace Cover Obstruction Database Access Functions .. 1719

<u>dbCreateInPlaceCoverObstruction</u>	1721
<u>dbDestroyInPlaceCoverObstruction</u>	1722
<u>dbGetInPlaceCoverObstructionAbstractFileName</u>	1723
<u>dbGetInPlaceCoverObstructionAbstractFilePath</u>	1724
<u>dbGetInPlaceCoverObstructionAbstractViewName</u>	1725
<u>dbGetInPlaceCoverObstructionBloat</u>	1726
<u>dbGetInPlaceCoverObstructionBlockageAttributeValue</u>	1727
<u>dbGetInPlaceCoverObstructionBlockageModel</u>	1728
<u>dbGetInPlaceCoverObstructionDoughnutHalo</u>	1729

Cadence Integrators Toolkit Database Reference

<u>dbGetInPlaceCoverObstructionLayers</u>	1730
<u>dbGetInPlaceCoverObstructionMaxMask</u>	1732
<u>dbGetInPlaceCoverObstructionMergeDistance</u>	1733
<u>dbGetInPlaceCoverObstructionMinMask</u>	1734
<u>dbGetInPlaceCoverObstructionNeedsRemodeling</u>	1735
<u>dbGetInPlaceCoverObstructionPinRemodeling</u>	1736
<u>dbGetInPlaceCoverObstructionPinCutOut</u>	1737
<u>dbGetInPlaceCoverObstructionSpacingModel</u>	1739
<u>dbInPlaceCoverObstructionExists</u>	1740
<u>dbResetInPlaceCoverObstruction</u>	1741
<u>dbSetInPlaceCoverObstructionAbstractFileName</u>	1742
<u>dbSetInPlaceCoverObstructionAbstractFilePath</u>	1743
<u>dbSetInPlaceCoverObstructionAbstractViewName</u>	1744
<u>dbIsInPlaceCoverObstructionCurrent</u>	1745
<u>dbSetInPlaceCoverObstructionBloat</u>	1746
<u>dbSetInPlaceCoverObstructionBlockageAttributeValue</u>	1747
<u>dbSetInPlaceCoverObstructionBlockageModel</u>	1748
<u>dbSetInPlaceCoverObstructionCurrent</u>	1749
<u>dbSetInPlaceCoverObstructionDoughnutHalo</u>	1750
<u>dbSetInPlaceCoverObstructionMergeDistance</u>	1751
<u>dbSetInPlaceCoverObstructionNeedsRemodeling</u>	1753
<u>dbSetInPlaceCoverObstructionPinCutOut</u>	1754
<u>dbSetInPlaceCoverObstructionPinRemodeling</u>	1756
<u>dbSetInPlaceCoverObstructionSpacingModel</u>	1757

19

Multi-Patterning Technology Functions 1759

<u>MPT Database Functions</u>	1760
<u>dbCellViewAreLayerShiftsValid</u>	1762
<u>dbCellViewClearLayerShifts</u>	1763
<u>dbCellViewHasLayerShifts</u>	1764
<u>dbCellViewUpdateLayerShifts</u>	1765

Cadence Integrators Toolkit Database Reference

<u>dbInstGetLayerShifts</u>	1766
<u>dbInstHasLayerShifts</u>	1768
<u>dbInstSetLayerShifts</u>	1769
<u>dbInstClearLayerShifts</u>	1771
<u>dbIsShapeColoringAllowed</u>	1772
<u>dbIsShapeColored</u>	1773
<u>dbGetShapeColor</u>	1774
<u>dbIsShapeColorLocked</u>	1775
<u>dbSetShapeColorLocked</u>	1776
<u>dbIsBlockageColored</u>	1777
<u>dbIsBlockageColoringAllowed</u>	1778
<u>dbGetBlockageColor</u>	1779
<u>dbSetBlockageColor</u>	1780
<u>dbGetViaLayer1Control</u>	1781
<u>dbGetViaLayer2Control</u>	1782
<u>dbGetViaCutLayerControl</u>	1783
<u>dbGetViaLayerControl</u>	1784
<u>dbIsViaColorStateLocked</u>	1786
<u>dbIsViaColorStateLayerLocked</u>	1787
<u>dbSetViaLayer1Control</u>	1789
<u>dbSetViaLayer2Control</u>	1790
<u>dbSetViaCutLayerControl</u>	1791
<u>dbSetViaLayerControl</u>	1792
<u>dbSetViaColorStateLocked</u>	1794
<u>dbSetViaColorStateLayerLocked</u>	1795
<u>dbSetViaColorInfo</u>	1797
<u>dbIsTrackPatternColored</u>	1799
<u>dbIsTrackPatternColorAlternating</u>	1800
<u>dbIsTrackPatternColoringAllowed</u>	1801
<u>dbGetTrackPatternFirstTrackColor</u>	1802
<u>dbSetTrackPatternFirstTrackColor</u>	1803
<u>dbSetTrackPatternColorAlternating</u>	1804
<u>dbCreateSameMaskGroup</u>	1805
<u>dbCreateDiffMaskGroup</u>	1806
<u>dbColoredShapeQuery</u>	1807
<u>dbGetIntegrationColorModel</u>	1809

Cadence Integrators Toolkit Database Reference

<u>dbGetColorModel</u>	1810
<u>dbSetIntegrationColorModel</u>	1811
<u>dbSetColorModel</u>	1812
<u>dbGetShapeColorLockType</u>	1813
<u>dbSetShapeColorLockType</u>	1814
<u>dbGetViaLayerColorLockType</u>	1816
<u>dbSetViaLayerColorLockType</u>	1818
<u>MPT Technology File Functions</u>	1820
<u>techGetIntegrationColorModel</u>	1821
<u>techGetLayerNumColorMasks</u>	1822
<u>techGetStdViaDefCutColoring</u>	1823
<u>techGetTechCutColoring</u>	1824
<u>techIsStdViaDefCutColoringSet</u>	1825
<u>techSetIntegrationColorModel</u>	1826
<u>techSetLayerNumColorMasks</u>	1827
<u>techSetStdViaDefCutColoring</u>	1828
<u>techSetTechCutColoring</u>	1829

20

CDBA Photonics Functions..... 1831

<u>Photonics Functions</u>	1832
<u>dbGetPhotonicPinFigAngle</u>	1833
<u>dbGetPhotonicPinFigRadius</u>	1834
<u>dbGetPhotonicPinFigWidth</u>	1835
<u>dbIsPhotonicsEnabled</u>	1836
<u>dbCellViewHasPhotonicPinFig</u>	1837
<u>dbSetPhotonicPinFigAngle</u>	1838
<u>dbSetPhotonicPinFigRadius</u>	1839
<u>dbSetPhotonicPinFigWidth</u>	1840
<u>dbUnsetPhotonicPinFigAngle</u>	1841
<u>dbUnsetPhotonicPinFigRadius</u>	1842
<u>dbUnsetPhotonicPinFigWidth</u>	1843

Cadence Integrators Toolkit Database Reference

<u>Curvy Core Functions</u>	1844
<u>Functions to Check Curvy Core Objects</u>	1846
<u>cclsBoundary</u>	1846
<u>cclsCurve</u>	1847
<u>cclsFacet</u>	1848
<u>cclsLine</u>	1849
<u>cclsMath</u>	1850
<u>cclsPath</u>	1851
<u>cclsPolyCurve</u>	1852
<u>cclsPolyCurveSegment</u>	1853
<u>cclsShape</u>	1854
<u>cclsSurface</u>	1855
<u>Functions to query object attributes</u>	1856
<u>ccGetCurveSpec</u>	1856
<u>ccGetFacetSpec</u>	1857
<u>ccGetFromFig</u>	1858
<u>ccGetPathBeginFacet</u>	1859
<u>ccGetPathBoundary</u>	1860
<u>ccGetPathCenterLine</u>	1861
<u>ccGetPathWidths</u>	1862
<u>ccGetPathEndFacet</u>	1863
<u>ccGetPolyCurve</u>	1864
<u>ccGetSurfaceBoundary</u>	1865
<u>Generator functions</u>	1866
<u>ccGenOAShape</u>	1866
<u>ccGenPolyCurveSegment</u>	1867
<u>ccGenSurfaceHole</u>	1868
<u>ccStartGenOAShape</u>	1869
<u>ccStartGenPolyCurveSegment</u>	1870
<u>ccStartGenSurfaceHole</u>	1871
<u>ccStopGen</u>	1872
<u>Miscellaneous Functions</u>	1873
<u>ccFreePathWidths</u>	1873
<u>ccReleaseObject</u>	1874
<u>Waveguide Functions</u>	1875
<u>Structs in the Waveguide Infrastructure</u>	1876

Enumerated Types and Enumerated Constants used in the Waveguide Infrastructure 1877

<u>techAllocWaveguideModePropArray</u>	1878
<u>techAllocWGDerivedShapeSpecArray</u>	1879
<u>techCreateWaveguideDef</u>	1880
<u>techDeleteWaveguideDef</u>	1882
<u>techFindWaveguideDef</u>	1883
<u>techFreeWaveguideModePropArray</u>	1884
<u>techFreeWGDerivedShapeSpecArray</u>	1885
<u>techGenWaveguideDef</u>	1886
<u>techGetWaveguideDefDerivedShapeSpecs</u>	1887
<u>techGetWaveguideDefLayer</u>	1888
<u>techGetWaveguideDefMinBendRadius</u>	1889
<u>techGetWaveguideDefMinWidth</u>	1890
<u>techGetWaveguideDefName</u>	1891
<u>techGetWaveguideDefPurpose</u>	1892
<u>techStartGenWaveguideDef</u>	1893
<u>techHasWaveguideDefMinBendRadius</u>	1894
<u>techSetWaveguideDefMinBendRadius</u>	1895
<u>techUnsetWaveguideDefMinBendRadius</u>	1896

21

Snap Pattern Definition Functions 1897

<u>techAllocEncSnappingLayerArray</u>	1899
<u>techAllocSnappingLayerArray</u>	1900
<u>techFreeEncSnappingLayerArray</u>	1901
<u>techCreateSnapPatternDef</u>	1902
<u>techCreateSnapPatternDefWithTrackGroups</u>	1904
<u>techFreeSnappingLayerArray</u>	1906
<u>techCreateSnapPatternDef</u>	1907
<u>techCreateSnapPatternDefWithTrackGroups</u>	1909
<u>techDeleteSnapPatternDef</u>	1911

Cadence Integrators Toolkit Database Reference

<u>techFindSnapPatternDef</u>	1912
<u>techFindSnapPatternDefByLPP</u>	1913
<u>techFindSnapPatternDefByName</u>	1914
<u>techFindSnapPatternDefByName</u>	1915
<u>techGenSnapPatternDef</u>	1916
<u>techGenSnapPatternDefForLPP</u>	1917
<u>techGetSnapPatternDefLayer</u>	1918
<u>techGetSnapPatternDefName</u>	1919
<u>techGetSnapPatternDefOffset</u>	1920
<u>techGetSnapPatternDefPurpose</u>	1921
<u>techGetSnapPatternDefSnappingLayers</u>	1922
<u>techGetSnapPatternDefStep</u>	1923
<u>techGetSnapPatternDefTrackGroups</u>	1924
<u>techGetSnapPatternDefTrackWidth</u>	1925
<u>techGetSnapPatternDefType</u>	1926
<u>techIsSnapPatternDefHorizontal</u>	1927
<u>techSnapPatternDefTypeToString</u>	1928
<u>techStartGenSnapPatternDef</u>	1929
<u>techStartGenSnapPatternDefForLPP</u>	1930
<u>techStringToSnapPatternDefType</u>	1931
<u>Example- Using Snap Pattern Definition Functions</u>	1932

22

Width Spacing Pattern Functions 1937

WSP Technology File Functions 1938

Miscellaneous Technology File Functions 1941

techMaskColorTypeToString 1941

techStringToMaskColorType 1942

Width Spacing Pattern Functions 1943

techAllocWidthSpacingPatternSpecArray 1943

techCreateWidthSpacingPattern 1944

techDeleteWidthSpacingPattern 1946

techFindWidthSpacingPattern 1947

techFreeWidthSpacingPatternSpecArray 1948

techGenWidthSpacingPattern 1949

Cadence Integrators Toolkit Database Reference

<u>techGenWidthSpacingPattern</u>	1950
<u>techGetWidthSpacingPatternAllowedRepeatMode</u>	1951
<u>techGetWidthSpacingPatternDefaultRepeatMode</u>	1952
<u>techGetWidthSpacingPatternName</u>	1953
<u>techGetWidthSpacingPatternOffset</u>	1954
<u>techGetWidthSpacingPatternPatterns</u>	1955
<u>techGetWidthSpacingPatternRepeatOffset</u>	1956
<u>techGetWidthSpacingPatternShiftColor</u>	1957
<u>techGetWidthSpacingPatternStartingColor</u>	1958
<u>techSetWidthSpacingPatternRepeatMode</u>	1959
<u>techStartGenWidthSpacingPattern</u>	1960
<u>Width Spacing Pattern Group Functions</u>	1961
<u>techCreateWidthSpacingPatternGroup</u>	1961
<u>techDeleteWidthSpacingPatternGroup</u>	1962
<u>techFindWidthSpacingPatternGroup</u>	1963
<u>techGenWidthSpacingPatternGroup</u>	1964
<u>techGetWidthSpacingPatternGroupName</u>	1965
<u>techGetWidthSpacingPatternGroupPatternNames</u>	1966
<u>techStartGenWidthSpacingPatternGroup</u>	1967
<u>Width Spacing Snap Pattern Definition Functions</u>	1968
<u>techCreateWidthSpacingSnapPatternDef</u>	1968
<u>techDeleteWidthSpacingSnapPatternDef</u>	1970
<u>techFindWidthSpacingSnapPatternDefByName</u>	1971
<u>techGenWidthSpacingSnapPatternDef</u>	1972
<u>techGenWidthSpacingSnapPatternDefForLPP</u>	1973
<u>techGetWidthSpacingSnapPatternDefDefaultActiveName</u>	1974
<u>techGetWidthSpacingSnapPatternDefLayer</u>	1975
<u>techGetWidthSpacingSnapPatternDefName</u>	1976
<u>techGetWidthSpacingSnapPatternDefOffset</u>	1977
<u>techGetWidthSpacingSnapPatternDefPatternGroupNames</u>	1978
<u>techGetWidthSpacingSnapPatternDefPatternNames</u>	1979
<u>techGetWidthSpacingSnapPatternDefPeriod</u>	1980
<u>techGetWidthSpacingSnapPatternDefPurpose</u>	1981

Cadence Integrators Toolkit Database Reference

<u>techGetWidthSpacingSnapPatternDefSnappingLayers</u>	1982
<u>techIsWidthSpacingSnapPatternDefHorizontal</u>	1983
<u>techStartGenWidthSpacingSnapPatternDefForLPP</u>	1984
<u>techStartGenWidthSpacingSnapPatternDef</u>	1985
Related Snap Pattern Functions	1986
<u>techAllocRelatedSnapPatternsSpecArray</u>	1986
<u>techCreateRelatedSnapPatterns</u>	1987
<u>techDeleteRelatedSnapPatterns</u>	1988
<u>techFindRelatedSnapPatterns</u>	1989
<u>techFreeRelatedSnapPatternsSpecArray</u>	1990
<u>techGenRelatedSnapPatterns</u>	1991
<u>techGetRelatedSnapPatternsName</u>	1992
<u>techGetRelatedSnapPatternsSpec</u>	1993
<u>techStartGenRelatedSnapPatterns</u>	1994
<u>Example- Using the WSP Technology File Functions</u>	1995
WSP Database Functions	2006
Miscellaneous Database File Functions	2010
<u>dbMaskColorTypeToString</u>	2010
<u>dbStringToMaskColorType</u>	2011
Global Grid Offset Functions	2012
<u>dbGetGlobalGridDefaultRepeatMode</u>	2012
<u>dbGetGlobalGridOffsetReferenceType</u>	2013
<u>dbGetGlobalGridRepeatMode</u>	2014
<u>dbSetGlobalGridDefaultRepeatMode</u>	2015
<u>dbSetGlobalGridOffsetReferenceType</u>	2016
Width Spacing Pattern Functions	2017
<u>dbAllocWidthSpacingPatternSpecArray</u>	2017
<u>dbCreateWidthSpacingPattern</u>	2018
<u>dbCreateWidthSpacingPatternWithRepeatMode</u>	2020
<u>dbDeleteWidthSpacingPattern</u>	2022
<u>dbFindWidthSpacingPattern</u>	2023
<u>dbFreeWidthSpacingPatternSpecArray</u>	2024
<u>dbGenWidthSpacingPattern</u>	2025
<u>dbGetWidthSpacingPatternAllowedRepeatMode</u>	2026
<u>dbGetWidthSpacingPatternDefaultRepeatMode</u>	2027
<u>dbGetWidthSpacingPatternName</u>	2028

Cadence Integrators Toolkit Database Reference

<u>dbGetWidthSpacingPatternOffset</u>	2029
<u>dbGetWidthSpacingPatternPatterns</u>	2030
<u>dbGetWidthSpacingPatternRepeatOffset</u>	2031
<u>dbGetWidthSpacingPatternShiftColor</u>	2032
<u>dbGetWidthSpacingPatternStartingColor</u>	2033
<u>dbStartGenWidthSpacingPattern</u>	2034
<u>Width Spacing Pattern Group Functions</u>	2035
<u>dbCreateWidthSpacingPatternGroup</u>	2035
<u>dbDeleteWidthSpacingPatternGroup</u>	2036
<u>dbFindWidthSpacingPatternGroup</u>	2037
<u>dbGenWidthSpacingPatternGroup</u>	2038
<u>dbGetWidthSpacingPatternGroupName</u>	2039
<u>dbGetWidthSpacingPatternGroupPatternNames</u>	2040
<u>dbStartGenWidthSpacingPatternGroup</u>	2041
<u>Pattern Region Functions</u>	2042
<u>dbGetPatternRegionCommittedPatterns</u>	2042
<u>dbGetPatternRegionActivePattern</u>	2043
<u>dbGetPatternRegionAllowedPatternGroups</u>	2044
<u>dbGetPatternRegionAllowedPatterns</u>	2045
<u>dbGetPatternRegionCreateType</u>	2046
<u>dbGetPatternRegionDefaultRepeatMode</u>	2047
<u>dbGetPatternRegionRepeatMode</u>	2048
<u>dbGetPatternRegionShiftColor</u>	2049
<u>dbHasPatternRegionActivePattern</u>	2050
<u>dbSetPatternRegionActivePattern</u>	2051
<u>dbSetPatternRegionAllowedPatternGroups</u>	2052
<u>dbSetPatternRegionAllowedPatterns</u>	2053
<u>dbSetPatternRegionDefaultRepeatMode</u>	2054
<u>dbSetPatternRegionRepeatMode</u>	2055
<u>dbShapeHasPatternRegionInfo</u>	2056
<u>dbGetPatternRegionName</u>	2057
<u>dbSetPatternRegionName</u>	2058
<u>dbGetPatternRegionCommittedPatternShiftColor</u>	2059

Cadence Integrators Toolkit Database Reference

<u>Width Spacing Snap Pattern Def Functions</u>	2060
<u>dbAllocSnappingLayerArray</u>	2060
<u>dbFreeSnappingLayerArray</u>	2061
<u>dbCreateWidthSpacingSnapPatternDef</u>	2062
<u>dbDeleteWidthSpacingSnapPatternDef</u>	2064
<u>dbFindWidthSpacingSnapPatternDefByName</u>	2065
<u>dbGenWidthSpacingSnapPatternDef</u>	2066
<u>dbGenWidthSpacingSnapPatternDefForLPP</u>	2067
<u>dbGenWidthSpacingSnapPatternDefForSnappingLPP</u>	2068
<u>dbGetWidthSpacingSnapPatternDefDefaultActiveName</u>	2069
<u>dbGetWidthSpacingSnapPatternDefLayer</u>	2070
<u>dbGetWidthSpacingSnapPatternDefName</u>	2071
<u>dbGetWidthSpacingSnapPatternDefOffset</u>	2072
<u>dbGetWidthSpacingSnapPatternDefPatternGroupNames</u>	2073
<u>dbGetWidthSpacingSnapPatternDefPatternNames</u>	2074
<u>dbGetWidthSpacingSnapPatternDefPeriod</u>	2075
<u>dbGetWidthSpacingSnapPatternDefPurpose</u>	2076
<u>dbGetWidthSpacingSnapPatternDefSnappingLayers</u>	2077
<u>dbIsWidthSpacingSnapPatternDefHorizontal</u>	2078
<u>dbStartGenWidthSpacingSnapPatternDef</u>	2079
<u>dbStartGenWidthSpacingSnapPatternDefForSnappingLPP</u>	2080
<u>Related Snap Pattern Functions</u>	2081
<u>dbAllocRelatedSnapPatternsSpecArray</u>	2081
<u>dbCreateRelatedSnapPatterns</u>	2082
<u>dbDeleteRelatedSnapPatterns</u>	2083
<u>dbFindRelatedSnapPatterns</u>	2084
<u>dbFreeRelatedSnapPatternsSpecArray</u>	2085
<u>dbGenRelatedSnapPatterns</u>	2086
<u>dbGetRelatedSnapPatternsName</u>	2087
<u>dbGetRelatedSnapPatternsSpec</u>	2088
<u>dbStartGenRelatedSnapPatterns</u>	2089
<u>Example- Using WSP Database Functions</u>	2090

23

Figure Set Functions 2103

FigSet Functions 2104

txAppendFig 2106

txAppendFigSet 2107

txClearFigSet 2108

txCloneFigSet 2109

txConcatFigSet 2110

txCopyFigSet 2111

txCreateFigSet 2112

txDeleteFigSet 2113

txFindFigSet 2114

txGetCellViewFigSets 2115

txGetFigSetBBox 2116

txGetFigSetCellView 2118

txGetFigSetEntry 2119

txGetFigSetLength 2120

txGetFigSetName 2121

txIsFigSet 2122

txMakeFigSet 2123

txMoveFigSet 2124

txPrependFig 2125

txPrependFigSet 2126

txRemoveFig 2127

txRemoveFigSet 2128

txRemoveFigSetEntry 2129

FigSetGroup Functions 2130

dbDeleteFigSetGroup 2131

dbFindFigSetGroup 2132

dbFindFigSetGroupByName 2133

dbGenFigSetGroups 2134

dbGetFigSetGroupName 2135

Cadence Integrators Toolkit Database Reference

<u>dbIsFigSetGroup</u>	2136
<u>dbMakeFigSetGroup</u>	2137
<u>dbStartGenFigSetGroups</u>	2138
<u>Example- Using the FigSet and FigSetGroup Code</u>	2139

A

<u>Dictionary of Basic CDBA Terms</u>	2143
<u>General Terms</u>	2144
<u>Terms Involving Actions</u>	2144
<u>Creating and Deleting Objects</u>	2145
<u>Actions Using Attributes</u>	2145
<u>Testing and Bookkeeping</u>	2146
<u>Actions Involving Input and Output for Cellviews</u>	2146
<u>Actions on Collections of Objects</u>	2146
<u>Adjectival Terms</u>	2147
<u>Alphabetical List of Abbreviations</u>	2147

Introduction to CDBA

This manual describes the functions in the Integrator's Toolkit: Database. The Integrator's Toolkit: Database is the Cadence C-level database access (CDBA) system.

To develop applications that use the Integrator's Toolkit: Database functions effectively, you must be an experienced C and UNIX programmer familiar with commonly used UNIX program development utilities, such as `vi` or `emacs`, `make`, `lint`, `cc`, and `dbx`, or the software development environment on your workstation.

All Integrator's Toolkit: Database (ITK) binaries must be used with the same corresponding Virtuoso Design Environment installation hierarchy. If they do not match—for example, a 6.1 ITK binary and a 5.2 DFII installation hierarchy—you can experience problems with data referenced from the installation hierarchy such as mismatched SKILL wrappers, context files, and environment settings.

This chapter introduces you to CDBA. CDBA is the C API layer used to access the OpenAccess database. CDBA is the means by which you can integrate your tools in a design framework. For more information, see the following sections:

- [Integrating Tools into a Framework](#) on page 84
- [Benefits of CDBA](#) on page 84
- [Supplementing CDBA Functions](#) on page 86

Integrating Tools into a Framework

In a framework environment, many tools are used together. These tools might not necessarily come from a single supplier. Cadence® strives to offer the best tool set, but this does not eliminate the need to integrate tools from other sources. These sources can include other tool vendors or internal development groups. Cadence offers a full set of products that allow users to integrate their tools into Cadence design framework II.

Design data or database integration is usually the first type of integration needed to make external tools work with the framework. This integration can often be accomplished through a netlist interface or through the Cadence SKILL extension language. At times, the only way to effectively integrate an external tool is with direct tool access to read and write the design data in the framework database.

Benefits of CDBA

CDBA routines provide nearly complete access to all the data stored in the OpenAccess database. You can create, modify, save, retrieve, and maintain data in the OpenAccess database. Cadence uses CDBA to design its own tools. CDBA functions are available at the programming level and allow tight and complete control over database operations. CDBA enforces a uniform data structure.

Because CDBA functions are built on a common model for data representation, the CDBA system provides a consistent methodology for creating electronic design data. This methodology is a central component of a framework design environment.

CDBA is a foundation for implementing a design process that you can extend and build upon when you use various applications. CDBA provides flexibility for a system to form an interface between the database and the internal data structures of a particular program.

CDBA allows many tools to use a common framework. CDBA provides a framework for integrating existing tools, serves as a foundation for developing new tools, and helps you to unify a set of tools. Different applications can share data effortlessly.

CDBA functions manage design data and provide abstraction from the mechanics of the data storage itself, so you can concentrate on the specific application you are working on.

However, because CDBA is more complex than most customer products, it requires careful thought to be put to its best use.

Database Types

This release of Design Framework II supports the OpenAccess database only. There is a function to verify the database type supported by the current executable.

dbGetDatabaseType

```
dbDatabaseType  
dbGetDatabaseType()
```

Description

Returns `dbcCDBADbType` or `dbcOpenAccessDbType`. The enum `dbcUnknownDbType` can be used to store the condition of not yet knowing the database type.

On this release, returns `dbcOpenAccessDbType`.

```
typedef enum dbDatabaseType {  
    dbcUnknownDbType      = 0,  
    dbcCDBADbType         = 1,  
    dbcOpenAccessDbType   = 2  
} dbDatabaseType;  
  
#define dbcMaxDatabaseType 2
```

Arguments

None.

Return Value

<code>dbcCDBADbType</code>	When called from a regular CDBA executable.
<code>dbcOpenAccessDbType</code>	When called from an OpenAccess executable.

Supplementing CDBA Functions

Although CDBA offers a complete set of data access functions, you might need to supplement CDBA functions with SKILL functions when you create data for tools in a design process.

Creating new data in the exact form required for an existing framework tool set is a more complicated process than reading data created by existing applications. Existing applications might have requirements for you to follow when you create data for tools in a design process.

Getting Started with CDBA

This chapter gives you the information you need to write CDBA functions. This includes the following topics:

- [CDBA Described](#) on page 88
- [Generator Functions](#) on page 89
- [Producer and Consumer Functions](#) on page 94
- [CDBA Naming Conventions](#) on page 95
- [Basics for Writing CDBA Functions](#) on page 101
- [Using the Virtuoso Design Environment](#) on page 106
- [Specifying Run-Time Options](#) on page 107
- [General Rules for Using CDBA](#) on page 108
- [Troubleshooting Your Programs](#) on page 108

CDBA Described

CDBA consists of procedural access layers on top of the OpenAccess database. CDBA insulates programs from changes made to the underlying physical data model. CDBA also takes advantage of the next generation of computing hardware and software environments.

CDBA functions include Component Description Format (CDF) functions and technology file functions. For more information on CDF functions, see the *Component Description Format User Guide*.

CDBA considers the following aspects of a good design:

- Robustness
- Performance
- Separate system and user address spaces
- Multiprocessing environment

Robustness

CDBA is as robust as possible without sacrificing performance. Objects and IDs are strongly typed, providing as much type checking as possible at compile time. Debugging facilities are designed into CDBA.

Performance

High performance is critical to a database for electronic design automation. To make CDBA run as fast as possible, most access functions are defined in a way that makes it possible to implement them as macros if needed.

Separate System and User Address Spaces

CDBA allows the database and your application to reside in separate process address spaces and communicate only via inter-process communication.

Multiprocessing Environment

CDBA is designed so that, in the future, the database and your application can run on two or more processors.

Example, Include, and Library Files to Use

Examples for using either CDBA or CDF functions are in

```
your_install_dir/itkDB/samples/DB
your_install_dir/itkDB/samples/CDF
```

The data structures and functions for CDBA and CDF are defined in

```
your_install_dir/itkDB/include/ITKDB.h
```

Link your application with

```
your_install_dir/itkDB/lib/ITKDB.a
```

Finding OpenAccess Shared Libraries

Your programs need to find the OpenAccess shared libraries. The recommended method is the following.

1. Install your programs in the appropriate directory.

```
❑ dfII/bin/32bit
❑ dfII/bin/64bit
as <program_name>
```

2. Install a symbolic link called *<program_name>* in the `dfII/bin` directory. The link should point to `../../../share/bin/cdnWrapperWithOA`.

The `cdnWrapperWithOA` script adds `QTDIR`, `tools/lib`, and the OpenAccess library path to `LD_LIBRARY_PATH` and calls the executable.

Generator Functions

Generator functions perform an action repeatedly on items, one by one, in a database that consists of either design data (OpenAccess) or technology data (technology file).

Sequence of Actions Used

Two functions usually work together to generate this sequence of actions: a function to start the generator (using the component `StartGen`) and a function to continue generating (using the component `Gen`) until all values have been acted on.

Together these two functions follow this sequence:

1. Initialize a particular generator.
2. Retrieve and act on each item in the file, one at a time. With each iteration of the generator function, a name and value for the item retrieved is returned.
3. Stop the generator.

Relationships Represented

Generator functions in CDBA represent one-to-many relationships in the logical data model. Through generators, you can visit every element of the relationship. These relationships can be direct or derived.

In a direct relationship, the objects are linked to a data structure such as a linked list. In a derived relationship, there are no direct pointers linking the objects, and the relationship is derived from other information, such as matching objects that have the same name.

For more detailed information on these types of relationships, see [“CDBA Database Objects”](#) on page 109.

Arguments in Generator Functions

In all generator functions, the first argument is always the generator state identifier returned from the `startgen` function. The rest of the arguments (ranging from one to many) are pointers to one or more user-defined variables used to return what is being generated. Arguments to generator functions are returned by means of indirection.

You supply pointers to objects of a given type as arguments. These pointers are updated by the generator function if it is successful. The pointers are invalid if the generator or function returns `FALSE`.

Values Returned

A generator function using pointers succeeds if all the following are true:

- The generator function can assign these user variables by indirection
- The variables are valid
- The variables can be safely referenced by the user

A returned value of `FALSE` signals that

- The values of the variables should not be used because they are undefined (never assigned and most likely contain garbage) or are invalid
- The generator is finished, with no more variables to assign values to

See [Chapter 12, “CDBA Relative Object Design Functions,”](#) for information on generator functions in technology files.

Naming Conventions for Generator Functions

Generator functions include the following components

- Prefix for application or interface used (such as `dd`, `db`, or `tech`)
- Sequence in the generation process
- Relationship between elements, if applicable

Sequence Indicated in Generator Functions

All generator functions use the following components:

- `StartGen`, indicating the function to prepare and start the generator
- `Gen`, indicating the function to generate
- `StopGen`, indicating the function to stop the generator

Relationship between Elements

The name of a generator function describes the elements of the one-to-many relationship it represents, and the actions between them.

In some generator functions, the relationship between the elements is written as *sourceToDest*, as in the function names `MasterToInst` or `LayerToPurpose`

When the one-to-many relationship starts from the cellview, you can omit both the *source* and the `To` connective. For example, instead of using the name `dbGenCellViewToMaster`, you can use the name `dbGenMaster`.

The Generator State

The style of generators is based on a generator state, which stores the internal state of each active generator in a data structure allocated and managed by the database. The user has complete control over starting, stopping, and stepping through multiple generators simultaneously.

Because generator states are allocated and are therefore always visible to CDBA, you can implement various checks to prevent users from undesirable interactions, such as deleting the next active element in the middle of a generation loop. When the internal structure of generator states is hidden from the user, only the following type definition is visible in the header file.

```
typedef dbGenState * dbGenStateId;
```

Starting the Generator

Each generator must be explicitly initialized by calling a specific `StartGen` function with the appropriate context information. The `StartGen` function returns a pointer to a newly allocated and initialized structure containing the generator state.

Example

```
dbGenStateId pState; /* generates all terms in a cellview */  
pState = dbStartGenTerm(cvId); /* starts the generator */
```

Generator Loop

After a generator is initialized, you can call the corresponding `Gen` function to get the relationship elements one at a time. A `Gen` function takes the generator state as its first argument. The argument is followed by pointers to formal variables used in the generator loop. If there are multiple formal variables, they must appear in the same order as they appear in the name of the generator.

The generator state is updated to reflect the new state of the generator loop. The generator function returns `FALSE` if there are no more elements in the relationship, and `TRUE` if there are more elements to generate. This makes it possible to implement a generator loop with a `while` statement.

Example

```
while (dbGenTerm(pState, &termId)) {  
    ...  
}
```

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

```
func(termId); /* loop through and process each element */
...
}
```

Stopping the Generator

At the end of a generator loop, you must call the `StopGen` function to notify CDBA that you are done with the generator. CDBA frees the generator state and invalidates the pointer to prevent its reuse. The `dbStopGen` function works for all generators.

Example

```
dbStopGen(pState);
```

Example

```
/* Using the shapes generator to generate all shapes on a */
/* given layer-purpose pair */
dbGenStateId pState; /* pointer to generator state */
dbShapeId shapeId; /* formal argument */
/* Initializes the generator */
pState = dbStartGenShape( cvId );
/* Uses a while loop to generate all shapes. */
while (dbGenShape( pState, &shapeId ) )
{
    /* processes shapeId */
    ...
    /* user can even pass generator state to another function */
    /* for further generation inside that function */
    func( pState, shapeId );
    ...
    break; /* user has complete control */
    ...
};
/* Inform the system that the generator is no longer active */
/* so the system can deallocate the generator state */
dbStopGen( pState );
```

Get Functions

Arguments to Get functions are returned by means of indirection. You supply pointers to objects of a given type as arguments. These pointers are updated by the Get function if it is successful. The pointers are invalid if the generator or function returns FALSE.

All Get functions return any parameters through the `propType` and `propValue` pointers. The parameter returned by `propValue` is either "evaluated" or "raw" data. Unless specifically

directed, all Get functions evaluate parameters before return. The "dumper" is the only utility or application that needs the unevaluated expression.

If you want the unevaluated parameter expression, use `techSetEvaluate (FALSE)` before calling the Get function of your choice. Remember to call `techSetEvaluate (TRUE)` when you are finished.

Producer and Consumer Functions

CDBA provides producer functions, which produce results processed by the consumer functions you provide. Each time you call a CDBA producer function, you must specify a pointer to your consumer function as one of the producer function arguments. For each item successfully produced, the producer calls your consumer function as the argument. This process is repeated until there are no more items to produce or the consumer returns FALSE to the producer.

Example

```
/* myConsumer( ) -
 *
 * This is a sample of a simple consumer function for the
 * dbProduceMemName( ) function. It prints the member
 * (such as member index) of the given full name and
 * the member name produced by the producer function.
 */

Boolean
myConsumer( memName, memIndex )
String      memName;
int         *memIndex;

{
printf( "Member %d is %s\n", memIndex, memName );
(*memIndex)++;
return(TRUE);
}

/* The following is a function that calls
 * dbProduceMemName( ) */
void
Afunc( )
{
Boolean      myConsumer( );
int          memIndex = 0;
dbProduceMemName( " (A,B,C) ", myConsumer, &memIndex);
}
```

CDBA Naming Conventions

Adoption of uniform naming conventions and standard terminologies for CDBA and application-specific names for functions, constants, or global variables are critical for making design framework II simple to use.

Components of Function Names

- Functions whose names include `Create` add sections to the technology file that do not exist, or replace entire or partial sections that do exist.
- Functions whose names include `Set` update existing data or add new data to existing sections of the technology file.
- Functions whose names include `Get` return data from sections of the technology file, or an error message if the data does not exist.
- Functions whose names include `Delete` remove a specified piece of data from a section of the technology file.

See [“Getting Started with CDBA”](#) on page 87 for more information on components of function names. See [“General Terms”](#) on page 2142 for a table on these components.

Components of Function Names

Construct a function name from related components. The combination of the components you use depends on what the function does.

General Rules

The following are some general guidelines for constructing function names:

- All names must be unique in the first 32 characters so that they are portable across systems.
- All names in CDBA, whether they are names of functions, constants, or global variables, begin with a package prefix and are composed of words connected in a predefined order. To avoid name conflicts with other packages, CDBA developers prefix `db` to every function and global variable name offered in CDBA.
- If you are an application developer, add a unique three-character package prefix to your function and global variable names to avoid name conflicts with CDBA function and global variable names.
- If the function implements a binary operation, then both the source and destination objects need to be specified. However, if the function implements a unary operation, you need to specify only one object or element.
- Function arguments are always given consistent, meaningful names that show their type and how they must be used.

The following function is a good model for this guideline.

```
dbCreatePolygon( cellViewId, layer, purpose,  
                nPoints,arrayOfPoints )
```

<code>cellViewId</code>	Tells <code>dbCreatePolygon</code> in which cellview to create the polygon.
<code>layer</code>	A scalar argument that defines on which layer to create the rectangle.
<code>purpose</code>	A scalar argument that defines which purpose to assign to the rectangle
<code>nPoints</code>	Specifies the number of vertices stored in <code>arrayOfPoints</code> .

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

`arrayOfPoints` Contains the vertices of the polygon to be created. The number of vertices is specified by `nPoints`.

Frequently Used Components

The following are frequently used components within names and examples for each. For a more complete list of components, see [“General Terms”](#) on page 2142.

Component Names	Examples
Package prefix	db, dm
Action verb	Create, Delete, Get, Set, Is
Qualifier for the source object	Empty, All
Type of source object	Shape, Net
Element within the source object	Layer
Connective	To
Type of destination object	Line, Polygon
Element within the destination object	nPoints
Condition being tested for (applies only to predicate functions)	Equal, Valid, Modified
Postfix, if any	Begin, End

Examples of Use of Components in Function Names

The following are examples of how these components are used with each other in a function name:

```
dbCreateRect
dbGetRectLP
dbGenLPToShape
dbIsRect
dbIsIdEqual
ilGetString
```

Components and Usage to Avoid in Function Names

To make names simple for users whose native language is not English, do not include the following in names:

- Articles (*a*, *an*, or *the*)
- Plurals (such as *layers* or *points*)
- Past tense forms of verbs
- Phonetic abbreviations such as 2 (as an abbreviation for *to*) or 4 (as an abbreviation for *four*)

Case Used in CDBA Names

All characters appear in lowercase except for the first character after each word boundary (such as `ilGetInt`).

The first character of a CDBA name, the package prefix, is always in lowercase. If you are an application developer, you must always capitalize the first letter of your prefixes to avoid naming conflicts with CDBA names.

Using Abbreviations in Function Names

To avoid unreasonably long names, abbreviate words longer than seven characters. Exceptions can be made when an easy-to-remember abbreviation does not exist.

The rules that apply to a single word within a name also apply to an abbreviation of a word within a name. A word or its abbreviation appears in lowercase except for the first character after a word boundary.

An abbreviation for multiple words within a name, however, can consist of all uppercase letters if each word is abbreviated to its first letter. For example, you can abbreviate *layer-purpose* to *LP*. If an official abbreviation exists for a word, use it in all appropriate names, regardless of the length of the name.

Naming Conventions for Function Arguments

The following sections describe types of function arguments.

Creation Functions

Arrange arguments for creation functions in the following order:

- Context arguments
- Attributes
- Argument interdependencies

Context

Specify context arguments in descending order of generality (from general to specific). In the following examples, `cellViewId` is the most general type of context argument.

```
dbCreateRect( cellViewId, ...)  
dbCreatePolygon( cellViewId, ...)
```

Attributes

For the object being created, specify attributes after the context arguments in descending order of importance. In the following examples, `layer` is a more important attribute than `purpose`.

```
dbCreateRect(..., layer, purpose, ...)  
dbCreatePolygon(..., layer, purpose, ...)
```

Argument Interdependencies

Specify descriptive arguments before the arguments they describe. The number of points always precedes the array of points it describes, as shown here:

```
dbCreatePolygon(cellViewId, layer, purpose, nPoints, pointArray)
```

This function also illustrates the arguments, attributes, and argument interdependencies in a creation function.

Retrieval Functions

Arguments in retrieval functions that return data using function arguments must be in the same order as when the object was first created. Examples are

```
dbGetPolygon( polygonId, &nPoints, &pointArray )  
dbGetPolygonLP( shapeId, &layer, &purpose )
```

Conventions for Prototype Functions

The following are conventions for describing arguments that appear in prototype functions:

- Pointers to data structures used to pass data to the function being called or to return data to the calling routine are indicated by adding the prefix `p` to the type name such as `pBBox`.
- When data is returned using the pointer, a comment follows the argument description (for example, `/* modifies BBox */`).
- Using a pointer replaces the old convention of preceding the argument with an ampersand (`&`). This old convention demonstrated normal usage, but left confusion about the argument type.

Return Value Conventions

Most database functions in CDBA return Boolean status values indicating whether their operations are successful. A few retrieval functions, however, do return data as values of the functions so that calls can be nested within other expressions. Many of these retrieval functions are implemented as macros: therefore, errors are caught at compile time rather than at run time.

Scalar values returned via function arguments are always copied over to your storage space. You must declare these variables in your own program and pass pointers to them in function calls.

Naming Conventions for Name Types

After a package prefix, there is an optional third lowercase character that describes the type of the name.

The following table shows the character and meaning corresponding to each name type.

Name Qualification Characters

Name type	Character	Meaning
Bit-field constants	b	A bit-field constant.
Constants	c	An enumerated constant.
Errors	e	The name of a structure describing an error.

Name Qualification Characters

Name type	Character	Meaning
Internal functions	i	An internal function. These functions must not be called directly by application programs.
Macros	m	A C-preprocessor macro.
Global variables	v	A public global variable.

Exceptions to Naming Conventions

Many existing property names were chosen when there was no system-wide, uniform naming convention. Because it is impractical for Cadence to change the names of existing properties, you need to learn historical names in addition to new names that conform to the uniform naming convention.

Rather than only following the new naming convention, keep applications as consistent as possible. If a different naming convention was originally used within an application, use the old convention to keep names consistent within the application. An example of an exception is the NLP (netlist property) prefix given to properties used by the netlister (because this prefix is in uppercase). To avoid unnecessary confusion for the user, give new properties for the netlister the same prefix.

Basics for Writing CDBA Functions

To write CDBA functions, you should know the following:

- Design framework II requirements
- Basic C data types used in CDBA
- Coding standards for writing CDBA functions

Design Framework II Requirements

Even though design framework II is written in portable C code, it does make basic assumptions about the compiler and the computer it runs on. The following assumptions must be true for a machine and its compiler before design framework II can run:

- Integers (`int`) are 32 bits wide.
- Long integers (`long`) are 32 bits wide.

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

- Short integers (`short`) are 16 bits wide.
- Characters (`char`) are 8 bits wide.
- Virtual address space must be at least 16 megabytes (MB).

Design framework II runs most efficiently when the following requirements are met:

- The amount of virtual memory available is at least 64 megabytes (MB).
- The amount of main memory available is at least 8 megabytes (MB) for applications using CDBA.

Design framework II can run correctly without these memory requirements, but the performance can become unacceptable.

Supported Platforms and Compilers for Integrator's Toolkit (itk)

The supported development platforms and C compilers are the following.

Platform	Compiler
Solaris Operating System 10 with support for STLPort4 C++ standard template libraries on x86_64 architecture	Solaris Compiler: Forte Developer/Studio 11 C++ 5.8
Solaris Operating System 10 with support for STLPort4 C++ standard template libraries on UltraSparc architecture	Solaris Compiler: Forte Developer/Studio 12 C++ 5.9
Linux (x86 and x86_64) Operating System: Redhat Enterprise 4.0	Linux Compiler: gcc 4.1.2
IBM Operating System: AIX 5.3	IBM Compiler: XL C++ 7.0
	When linking on AIX 5.3: Do not use <code>-bexpall</code> or <code>-bexpfull</code> . Always link with the Tcl shared library <code>libTcl8.4</code> . So instead of <code>libTcl8.4.a</code> . Place <code>-lm -lc</code> in front of <code>-ltaDesign</code> .

C Data Types Used in CDBA

To simplify programming and improve code readability, the following data types are predefined in `db.h`, the header file that must be included in every CDBA program.

Boolean

A Boolean specifies that the variable or function can only take the value of one of two predefined constants.

- `TRUE` (nonzero)
- `FALSE` (zero)

String

Strings specify that the variable or function is of type `(char *)`, a pointer to a character string. A NULL string is not equivalent to an empty string.

FuncIntPtr

`FuncIntPtr` declarations specify that the variable or function indicates a pointer to a function that returns an integer. This type is useful for passing functions as arguments to other functions, such as when handler functions are registered.

FuncBooleanPtr

`FuncBooleanPtr` specify that the variable or function points to a function that returns a Boolean value. This type is useful for passing functions as arguments to other functions, such as when handler functions are registered.

Enumerated Types and Enumerated Constants

Variables that can only take on one of a limited number of values are declared in the C programming language as enumerated types (`enums`) whenever possible. You can also use the `#define` construct to define enumerations as unsigned integer constants.

All enumerated constants begin with the prefix `dbc`, and enumerated constants that belong to the same enumerated type are often grouped together using the same root words. Typically, an enumerated type has an unknown value, which is returned in some error conditions.

Examples

The following are examples of enumerated types:

```
#define dbcConstantOne 1
#define dbcConstantTwo 2
enum dbPurpose {
    dbcUnknownPurpose,
    dbcDrawingPurpose,
    dbcWirePurpose,
    dbcPinPurpose
};
typedef enum dbPurpose dbPurpose;
```

Coding Standards for CDBA Functions

Coding standards exist for code in general, for header files, and for calls to macros.

General Coding Standards

To write C programs that work in CDBA, follow these coding standards:

- Use only reentrant code. Because most commands can be nested, not using reentrant code can possibly lead to recursive calls on the same functions.
- Do not use global and static variables.
- Begin function names and global variables with a unique three-character prefix to avoid name conflicts. Use three-character prefixes in your applications to avoid name conflicts with Cadence packages, which often use two-character prefixes. The first character must be uppercase.
- Use a default case to catch all unexpected values in every switch statement, particularly ones that switch on an enumerated type. This default case is extremely important to keep the code from breaking when new enumerated types are added.
- Do not rely on the integer value of one enumerated constant being greater than the value of another enumerated constant, unless explicitly permitted by documentation in the manual.
- Do not directly address the members in a structure, unless this is explicitly permitted by documentation in the manual.
- Within a structure, declare all unions that contain a tag member, in addition to the union member. The tag member must be an enumeration of the union members, plus the special constant NULL. The value of NULL must be 0, which indicates an un-initialized structure. Set the tag member whenever the value is set.

The following is an example of this.

```
struct pkgUnion {
enum {
    pkgcNull = 0
    pkgcIntType = 1
    pkgcFloatType = 2
    pkgcBooleanType = 3
} tag;
union {
    int      aInt;
    float    aFloat;
    char     aBoolean;
} value
};

typedef struct pkgUnion pkgUnion;
```

- Do not call any UNIX system function that can interfere with the operation of the graphics editor. In addition to all system functions that deal with signals and graphics, avoid using system functions (such as `sbrk`, `setjmp`, and `longjmp`) that affect the operating environment and can potentially have adverse effects on Cadence software.
- Do not use the `stdin`, `stdout`, and `stderr` standard file descriptors for input and output. Use functions provided by CDBA for getting interactive user input and for displaying prompts and messages.
- Install all binary programs built on top of CDBA in the standard Cadence release hierarchy, just as you install the Cadence binary programs. The proper operation of CDBA depends on certain standard directories and files, which are stored in the Cadence release hierarchy.

Header Files

To use the C language effectively, Cadence provides the header file (`itkDB.h` for CDBA) containing internal data structure declarations, as well as constant definitions. Because of the way DFII is implemented, the contents of header files can change in the future. Do not write your programs based on undocumented details exposed by this header file. Always refer to this manual for information about CDBA.

Macros and Functions

CDBA allows most database access routines to be implemented as either a function or a macro. For programming ease, most of the macros in CDBA are implemented so they can be called like a function, without any undesirable side effects. Macros without undesirable side effects have the same `db` prefix as functions. Therefore, you do not have to distinguish between these macros and any functional counterparts.

The following are special considerations and good practices in using macros and functions:

- A performance-critical routine can occasionally require so much code that it is desirable to implement the routine both as a macro and as a function. Thus, you can choose between the two implementations according to the needs of your application. In this situation, the macro implementation must be given the prefix `dbm` to distinguish it from the function implementation.

In only a few cases, you must distinguish between a macro and a procedure that have the same functionality. This distinction is necessary because procedure calls can take up too much execution time, and macro calls can take up too much code space.

An additional prefix letter, `m`, is added to the name of the macro after the package prefix `db` (as in `dbm`) so you can choose between the two implementations.

- In the C programming language, macros can be defined so their arguments appear many times within the macro expansion. Therefore, it is a bad programming practice to put expressions with permanent side effects (such as *assignments*) in the argument list of a macro call. Never pass as arguments to macros any expressions that have side effects.

Even when you use expressions with no permanent side effects, the processing of the macro is slowed by repeated evaluations of the same expression, and therefore, the macro can expand into a larger amount of in line code.

- If a macro refers to any of its arguments more than once and is therefore vulnerable to side effects, you must give it the prefix `dbm` to alert the user that it is such a macro. Use the prefix even when the macro has no functional counterpart.
- It is always good practice to first assign the value of an expression to a variable, then to use that variable in the argument list of a macro. If a routine is documented as being implemented as a macro, you must refrain from calling it with any expression in the argument list. The argument lists in your macro calls must contain only variables or simple expressions.

Using the Virtuoso Design Environment

Include the header file `itkDB.h` in every source file that refers to one of the Cadence Integrator's Toolkit: Database (ITKDB) routines.

You can access the Virtuoso Design Environment routines through one of the library archives in the `your_install_dir/dfII/lib` directory. All Virtuoso procedures do intensive checking to help find programming inconsistencies quickly.

Include the `itkDB.a` file in the list of library archives given to the loader when linking an executable binary for your program. The order of libraries is important. The `itkDB.a` library must appear after all libraries that refer to ITKDB routines. Otherwise, the loader is not able to link your program successfully.

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

Note: The `itkDB.a` archive library depends on the `libplotCommon.so` shared library, which has a dependency on `freetype.so`. Applications that compile with `itkDB.a` must therefore, link with `freetype.so`. For example, the `itkDB` sample makefiles link the samples with `freetype.so`.

If you build your application with `itkDB.a`, license 111 must be present to run the application. License 111 is checked out every time you invoke any application from within the Virtuoso tool suite. This ensures that a valid user is trying to run the application.

If license 111 is not present, the application issues the following message and exits.

```
dbInit: Existence of Cadence(R) Design Framework II license (product 111) is
required to execute ITKDB based program.
```

In advanced nodes version, `itkDB.a` will also checkout license 95111 along with license 111.

ICADV20.1 Only

In advanced nodes, `Virtuoso_Adv_Node_Framework` license must be present in addition to license 111 to run an application. If either of these licenses is not present, the application issues an error message and exits.



Your programs and libraries must have function names different from those used by Cadence. If name conflicts do exist, the loader cannot successfully link your program, or the loader might produce an executable program that behaves erroneously.

Refer to the standard documentation that comes with your Cadence programs for details.

Specifying Run-Time Options

You can specify several processing options for your program. For example, if your program has the name `myProg`, the command line to run your program is

```
myProg [-dbAssert] [-dbCheck] [-dbBind] your_options
```

`-dbAssert`

Triggers an intensive checking of various conditions. If any internal consistency checks are not met, the program automatically stops processing. You can look at the log file (that is, `~/CDS.log`) to find out which condition failed.

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

<code>-dbCheck</code>	Causes the database code to extensively check database consistency at predetermined intervals. Warnings are issued when the database is inconsistent, but processing does not stop.
<code>-dbBind</code>	Opens and binds the master of each instance to the corresponding instance whenever a cellview is read in. With this option, you can traverse down a hierarchy without explicitly calling routines that bring master cellviews into virtual memory.
<code>your_options</code>	Modifies the <code>dbInit</code> function when CDBA command line options are present. After the call to <code>dbInit</code> , only the program name and options not recognized by CDBA remain in the command line: <code>myProg your_options</code>

For more information on run-time options, refer to [“CDBA Input and Output Functions”](#) on page 161.

General Rules for Using CDBA

Cadence carefully documents all features of CDBA in this manual. You must use these features only as documented. Do not use undocumented features or try to take shortcuts when using design framework II.



Do not access database objects directly. Always use the CDBA routines. There are many interdependencies, and the underlying data structures can change without notice.

Troubleshooting Your Programs

Even when you write your program carefully, it still might not work correctly. In addition, the source of the problem might not be obvious if there are many interactions with the database. Under these circumstances, take some or all of these steps to isolate the problem and identify obvious errors. The following checklist can be helpful.

- Is the code compiled and linked without errors? If the compiler does not accept a call to a database routine, check the documentation for that call to make sure you do not have any programming errors.

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

- Have any run-time errors occurred, such as a bad ID return from a `dbOpenCellViewByType` call? This can happen if you do not have write permission on a cellview and are trying to open it for writing or appending.
- Have you checked the log file for debugging and error messages? The log file (that is, `~/CDS.log`) normally contains a complete log of run-time errors.
- Have you used a symbolic debugger to trace the problem? With a symbolic debugger, you can stop your program and examine the variables examined in the middle of a run. This interactive querying is often helpful in determining the status of a program.
- Is your storage allocation safe? Although this interface will probably change in the future, setting the shell environment variable `CDS_MEM_DEBUG` to a value of `global:GET;PUT;PAD` can help you spot this type of problem. Examine your code for reuse of storage that belongs to CDBA. The following code fragment overwrites the name string returned from CDBA:

```
String name;  
name = dbGetCellViewCellName(cellViewId);  
printf(strcat(name, " opened."));
```

- Are the proper actions being taken upon return from a database call? For instance, if the database call returns an error, does the program handle it properly, or does it continue running without checking for error conditions? In the previous example, the return value from `dbGetCellViewCellName` must be compared to `dbcNullString` before use elsewhere.

Cadence Integrators Toolkit Database Reference

Getting Started with CDBA

CDBA Database Objects

Design data is stored as objects on the OpenAccess database. Each type of object represents a distinct concept in the world of electronic design automation. Examples of objects are rectangles, terminals, instances, and cellviews.

The OpenAccess database stores both physical and logical information about a design. Physical information is stored in objects such as geometrical shapes that directly correspond to their physical implementations. Logical information, on the other hand, is stored in objects such as nets that can exist without any corresponding physical realizations.

For more specific information on database objects, see these sections:

- [Object Identifiers](#) on page 112
- [Object Types and Classes](#) on page 112
- [Attributes](#) on page 114
- [Properties](#) on page 115
- [Attributes and Properties Compared](#) on page 115
- [Property Bags](#) on page 116
- [Relationships among Objects](#) on page 121
- [Objects and IDs](#) on page 123

Object Identifiers

To access an object in the database, you must first obtain a handle to the object known as the object identifier, or ID. IDs are data structures containing information that uniquely identifies the database objects the data structures refer to. The contents of IDs can only be created by database routines. End users cannot alter them directly. For all functions that operate on database objects, you need to specify the ID as a function argument.

This section discusses a generalized concept of database objects rather than the explicit CDBA data structures.

Object Types and Classes

Each object belongs to a type, and each type contains a number of attributes that describe the object. When a number of distinct object types share enough attributes to be more easily referred to as a single class, you create a data-type abstraction called an object class for these object types. The attributes of the object types, not the object types themselves, determine the grouping.

The different object types and classes form a class hierarchy. At the top of the hierarchy is the class of all database objects. At the bottom of the hierarchy are leaf nodes that represent an object type you can create, delete, and save 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.

The following table, [Database Objects and IDs](#) on page 112, shows a list of object classes, the names of the corresponding enumerated constants, and the names of corresponding IDs. The enumerated type `dbType` represents all enumerated constants representing object types.

Database Objects and IDs

Object class	Enumerated constant	Name for ID of object	Name for ID of enumerated constant
cellview	dbcCellViewType	dbCellViewId	dbcNullCellViewId
figure	-	dbFigId	dbcNullFigId
shape	-	dbShapeId	dbcNullShapeId
rectangle	dbcRectType	dbRectId	dbcNullRectId
polygon	dbcPolygonType	dbPolygonId	dbcNullPolygonId

Cadence Integrators Toolkit Database Reference

CDBA Database Objects

Database Objects and IDs, *continued*

Object class	Enumerated constant	Name for ID of object	Name for ID of enumerated constant
line	dbcLineType	dbLineId	dbcNullLineId
path	dbcPathType	dbPathId	dbcNullPathId
arc	dbcArcType	dbArcId	dbcNullArcId
ellipse	dbcEllipseType	dbEllipsoid	dbcNullEllipsoid
donut	dbcDonutType	dbDonutId	dbcNullDonutId
dot	dbcDotType	dbDotId	dbcNullDotId
label	dbcLabelType	dbLabelId	dbcNullLabelId
any instance	-	dbAnyInstId	dbcNullAnyInstId
instance	dbcInstType	dbInstId	dbcNullInstId
mosaic instance	dbcMosaicInstType	dbMosaicInstId	dbcNullMosaicInstId
mosaic	dbcMosaicType	dbMosaicId	dbcNullMosaicId
signal	dbcSigType	dbSigId	dbcNullSigId
net	dbcNetType	dbNetId	dbcNullNetId
terminal	dbcTermType	dbTermId	dbcNullTermId
pin	dbcPinType	dbPinId	dbcNullPinId
instance header	dbcInstHeaderType	dbInstHeaderId	dbcNullInstHeaderId
instance terminal	dbcInstTermType	dbInstTermId	dbcNullInstTermId
instance pin	dbcInstPinType	dbInstPinId	dbcNullInstPinId
property	dbcPropType	dbPropId	dbcNullPropId
property bag	dbcBagType	dbBagId	dbcNullBagId
group	dbcGroupType	dbGroupId	dbcNullGroupId
group member	dbcGroupMemType	dbGroupMemId	dbcNullGroupMemId
layer	dbcLayerType	dbLayerId	dbcNullLayerId
display	dbcDisplayType	dbDisplayId	dbcNullDisplayId

Cadence Integrators Toolkit Database Reference

CDBA Database Objects

Database Objects and IDs, *continued*

Object class	Enumerated constant	Name for ID of object	Name for ID of enumerated constant
text display	dbcTextDisplay	dbTextDisplayId	dbcNullTextDisplayId
zero value for generators			dbcNoMoreObj

Attributes

Each object type has a predefined set of attributes that can be set and retrieved using CDBA functions. The three types of attributes are

- Mandatory
- Optional
- Derived

Mandatory Attributes

Most attributes are mandatory. You must specify mandatory attributes at the time the object is created.

Optional Attributes

Some objects can have optional attributes. You can add optional attributes after an object is created.

Derived Attributes

Derived attributes are reconstructed from other object attributes. You have read-only access to derived attributes. Derived attributes are not stored in the database, but they play an important role in speeding up repetitive computation.

Properties

Properties store information about objects and serve as a mechanism for extending the basic framework of the database. A property has a name (which is a string) and a value. A property value can be a Boolean value, an integer, a floating-point number, a string, or an integer representing time. You can attach an arbitrary number of properties to an object.

Attributes and Properties Compared

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 information currently stored as properties can be so essential to system operation that to ensure its robustness and speed, the database itself needs to manage this information directly. On the other hand, information currently stored as attributes can be so rarely used that the attributes should be moved out of the database layer into the application layer.

Mandatory attributes are stored directly in data structures that are optimized for storage efficiency and performance. Less frequently used optional attributes are better stored as “properties” instead of as part of the basic data structures, because properties take up storage space only when they are present. If you redefine optional attributes as properties, the optional attributes become visible to the user as normal properties.

Although properties are not mandatory for the database, some information currently stored as properties is essential to an application’s operations. Therefore, properties are mandatory for an application.

Because mandatory information stored as a property now might eventually be stored as an attribute later, some minor incompatibilities between different versions of CDBA are inevitable.

Currently, there is no solution to the problem, but the following coding conventions might minimize incompatibilities:

- Avoid name conflicts between attributes and properties that Cadence might introduce in the future. Adopt a naming convention that does not conflict with the Cadence naming convention.

A simple convention is to make the names of all your properties begin with an uppercase letter (except “X,” which signifies the X Window System) because the Cadence

convention is to begin names with a lowercase letter. An even safer convention, however, is to add to all your property names a unique three-letter prefix identifying your company.

- Although an attribute might be stored as a property, continue to treat it as you do other attributes. Always use the CDBA functions that are provided for manipulating that attribute.

Property Bags

Properties of OpenAccess objects can be accessed through property bags.

Storage Hierarchy Structure

Access to properties is handled through a concept called a property bag. The property bag is a virtual space that contains only properties and is managed by the procedural interface (PI). Each library, cell, or cellview can have an associated property bag. The PI keeps track of properties and manages interaction with properties.

Property Bag Access

You manipulate library, cell, and cellview properties externally through the same PI you use in DFII. When you create, access, and update a library, cell, or cellview property, this action causes an implicit I/O on the property bag, unless you call the property bag explicitly. The implicit I/O also supports transparent migration.

Opening a design cellview does not trigger the opening of the property bag. You open the property bag only when you need to. You can also open the property bag without also opening a design.

The following summarizes I/O implicit interaction with properties for both access and internal editing.

- When you access library, cell, or view properties without doing an explicit I/O on the property bag, the I/O implicitly does the following.
 - ❑ Opens the property bag for `read`.
 - ❑ Gets the property requested, if any.
- When you edit library, cell, or cellview properties without doing an explicit I/O on the property bag, the following occur implicitly.
 - ❑ The I/O opens the property bag for `edit`.

- ❑ You edit the property.
- ❑ You save and reopen the bag for `read`.

Therefore, each edit implies a pair of open and save operations. If you are editing more than one property in a given property bag, you can reduce the I/O overhead by opening the bag explicitly, making your edits, then saving and closing the bag.

A property bag residing in virtual memory (VM) might be out of date in a multiuser environment. Therefore, if you need to get the most current version, you must use `dbMakeBagCurrent` prior to any access or edit.

Explicit I/O Functions on Property Bags

If you open a property bag explicitly and access, create, and update properties in the property bag, the behavior enforced for the property bag is the same as the one for the design cellview. There is no I/O or locking overhead until you save, close, or purge the property bag.

Explicit I/O functions on property bags follow, along with brief explanations of their options.

dbOpenBag

```
dbBagId  
dbOpenBag( ddId id, String mode  
           );
```

Opens the library, cell, or view property bag in the given mode. Valid modes are `r` for read, and `a` or `w` for append. Locking and checking out policies are identical to `dbOpenCellViewByType`.

Opening the property bag for append or write locks out other users from updating the same property bag until the property bag is purged or reopened for read. Internally a reference count (`refCount`) is updated every time a property bag is opened and closed. A write count (`wCount`) is also updated every time a property bag is opened and closed for editing.

dbCloseBag

```
Boolean  
dbCloseBag( dbBagId bagId, String mode  
           );
```

Decrements the property bag `refCount`. If `refCount` becomes zero, the `dbCloseBag` function purges the property bag `refCount` from virtual memory and unlocks the property

Cadence Integrators Toolkit Database Reference

CDBA Database Objects

bag. Closing for edit decrements the write count. If the write count reaches zero but the `refCount` is still greater than zero, the property bag is downgraded to read mode and other users can edit it. Valid modes are `r` for read, and `a` or `w` for append.

dbGetRefCount

Boolean

```
dbGetRefCount( ddId id, String mode
               );
```

For DFII on OpenAccess, returns a value that is one less than the OpenAccess cellview `refCount` value that is returned by `oaCellView::getRefCount()`. This applies to cellviews that represent instance sub-masters. Exceptions are instance super-master cellviews.

dbPurgeBag

Boolean

```
dbPurgeBag( dbBagId bagId
            );
```

Purges the property bag from virtual memory (VM) and releases the lock applied on the bag.

dbSaveBag

Boolean

```
dbSaveBag( dbBagId bagId
           );
```

Writes the VM image of the property bag.

dbRefreshBag

Boolean

```
dbRefreshBag( dbBagId bagId
              );
```

Refreshes the VM image of the property bag with the stored image. If no stored image exists and the property bag is in append mode, it resets the property bag to empty (the VM image is truncated).

dbReOpenBag

Boolean

```
dbReOpenBag( dbBagId bagId, String mode
             );
```

Reopens the property bag for the given mode. Valid modes are `r` for read, and `a` or `w` for append.

dbGetBag

dbBagId

```
dbGetBag( ddId id
          );
```

Returns the property bag ID of the given `ddId`. Opens and reads the property bag if it is not yet in VM. If the property bag does not exist, it returns NULL ID. It does not increment the `refCount` if the property bag is already in VM.

dbMakeBagCurrent

Boolean

```
dbMakeBagCurrent( dbBagId bagId
                  );
```

Refreshes the given property bag if it is out of date relative to the saved version. Returns FALSE only if it finds an error.

dbGetBagDdId

ddId

```
dbGetBagDdId( dbBagId bagId
              );
```

Returns the `ddId` associated with the property bag.

dbFindOpenBag

dbBagId

```
dbFindOpenBag( ddId id
               );
```

If the bag for the given `ddId` exists in VM, `dbFindOpenBag` returns it. Otherwise, it returns NULL.

Information Retrieval Functions on Property Bags

The following functions retrieve information about various characteristics of property bags.

dbIsBagId

```
Boolean  
dbIsBagId( dbBagId bagId  
           );
```

Returns TRUE if the given `bagId` is valid.

dbGetBagTimeStamp

```
time_t  
dbGetBagTimeStamp( dbBagId bagId  
                  );
```

Returns the `timeStamp` of the property bag. The property bag `timeStamp` is not set or modified until you update the property bag.

dbIsBagCurrent

```
Boolean  
dbIsBagCurrent( dbBagId bagId  
               );
```

Returns TRUE if the property bag is current relative to the saved version.

dbIsBagReadable

```
Boolean  
dbIsBagReadable( ddId id  
                );
```

Returns TRUE if the property bag exists for the given `ddId` and the user has read access on the bag. The property bag must exist.

dbIsBagWritable

Boolean

```
dbIsBagWritable( ddId id  
                );
```

Returns the UNIX file permission of the property bag regardless of the open mode of the property bag. For example, `basic` is a system owned library, `dbIsBagWritable` will always return `FALSE` on its bag. If you own a library, and its property bag has write permission, even if the bag is opened in `r` mode, `dbIsBagWritable` will still return `TRUE`. A new property bag that exists only in VM is not considered writable.

The following `db` functions

- Accept `dbId`, `ddId` and `dbBagId`
- Open, save, and reopen the property bag for an accepted `ddId`, if applicable

```
dbCreateProp(),  
dbCreateHierProp(),  
dbCreateRangeProp(),  
dbCreateEnumProp();  
dbReplaceProp(),  
dbReplaceRangeProp(),  
dbReplaceHierProp(),  
dbReplaceEnumProp(),  
dbReplacePropSpecial();  
dbGetPropByName();  
dbStartGenProp();  
dbGenProp();  
dbObjHasProp();  
dbCopyProp();  
dbMergeProp();  
dbCopySingleProp();  
dbMergeSingleProp();
```

Relationships among Objects

The OpenAccess database stores a large number of predefined relationships among different object types for efficiency and consistency. Such relationships are typically hidden from you, but you must explicitly create and maintain other object relationships. Some predefined relationships are

- Mandatory relationships

- Optional relationships
- Direct relationships
- Derived relationships
- One-to-one relationships
- Many-to-one relationships
- One-to-many relationships
- Relationships between logical and physical objects

Mandatory Relationships

Mandatory relationships are required relationships in the database. For example, every terminal must be attached to an electrical net. Therefore, you cannot create a terminal without specifying a net to which it connects.

Mandatory relationships are very often unidirectional. For example, you might have a net that does not connect to a terminal.

To maintain database consistency, when you delete an object belonging to a mandatory relationship, other objects belonging to that relationship are also deleted, because they cannot exist on their own. For example, if an electrical net is deleted, the terminal attached to that net is deleted as well because it cannot exist on its own.

Optional Relationships

Optional relationships in the database are those which are not required but which you create. You can create objects independently, then create the relationships among the objects.

Functions are provided for adding objects to and subtracting objects from the relationship. An example of a function expressing an optional relationship is

```
dbAddFigToNet( figId, netId )
```

Direct Relationships

Direct relationships have a one-to-one correspondence to linkage pointers within the database.

Derived Relationships

Derived relationships must be reconstructed each time their corresponding access functions are called, because derived attributes are not stored in the database. It is therefore usually much faster to access direct relationships than to access derived relationships.

One-to-One and Many-to-One Relationships

When a predefined relationship is a one-to-one or a many-to-one relationship, an access function allows the user to go from the source object to the related object. For example, the user can go from an instance to its master using the `dbGetInstMaster` function.

One-to-Many Relationships

When a predefined relationship is a one-to-many relationship, use the generator function to traverse the relationship and visit the related objects. For example, the user can go from a master to all its instances using the `dbGenMasterToAnyInst` generator function.

Relationships Between Logical and Physical Objects

When a logical object has a corresponding physical realization represented by one or more physical objects, the objects are linked together in the database. Access functions go from a logical object to its corresponding physical objects and vice versa.

An example of a function that accesses a physical object (a figure) from a logical object (a net) is

```
dbGenNetToFig(...)
```

Objects and IDs

The API is strongly typed so that the compiler catches as many semantic errors as possible at compile time. Compared to the alternative of a single generic C type for all database IDs, which performs type checking at execution, a strongly typed API facilitates error checking much earlier, is more efficient, and is inherently more robust. It does, however, occasionally require explicit typecasting among different types of IDs.

Conversions between IDs

All conversions between compatible ID types have to be explicit using the C typecasting mechanism. You must take special care not to cast a generic ID into a specific ID, until you know that the ID is indeed pointing to an object of that particular type. Casting an ID into an incompatible ID type creates a fatal programming error. You can reliably detect it only at run time, using the debug version of CDBA.

Functions to Retrieve Object and ID Information

Use the following functions to retrieve information about objects and IDs.

dbGetType

```
dbType
dbGetType( dbId id
          );
```

The `dbGetType` function returns an enumerated constant, which represents the object type referred to by the ID. For example, use `dbcRectType` if the object is a rectangle. `dbGetType` is currently implemented as a function but might be implemented as a macro in the future.

dbIsIdEqual

```
Boolean
dbIsIdEqual( dbId id1, dbId id2
            );
```

The `dbIsIdEqual` macro returns `TRUE` if `id1` and `id2` refer to the same database object. Always use this macro to test for the equality of two IDs instead of relying on the C operator `“==”`.

The following set of functions are fast, safe, and quiet. They never produce a core dump, no matter what arguments are passed to them, and they never generate error messages.

dbIsId

```
Boolean
dbIsId( dbId id
       );
```

The `dbIsId` function returns TRUE if the argument is a valid database object ID. If not, it returns FALSE.

dbIsCellViewId

Boolean

```
dbIsCellViewId( dbCellViewId id  
                );
```

The `dbIsCellViewId` function returns TRUE if the argument is a valid cellview ID. If not, it returns FALSE.

dbIsFigId

The `dbIsFigId` function is documented in [“CDBA Figure Functions”](#) on page 205.

dbIsShapelId

The `dbIsShapelId` function is documented in [“Shape Figure Functions”](#) on page 410.

dbIsAnyInstId

Boolean

```
dbIsAnyInstId( dbAnyInstId id  
               );
```

The `dbIsAnyInstId` function returns TRUE if the argument is a valid instance or mosaic instance ID. If not, it returns FALSE.

Cadence Integrators Toolkit Database Reference

CDBA Database Objects

CDBA Representation of Elements

This chapter covers the spacial and geometrical elements of a display, and the C functions used to implement them.

For more information, see the following topics:

- [Database Units and User Units](#) on page 128
- [Coordinates](#) on page 132
- [Points](#) on page 132
- [Bounding Boxes](#) on page 134
- [Angles](#) on page 140
- [Shape Transformation](#) on page 141
- [Distance, Direction, and Time](#) on page 143

Database Units and User Units

A database unit is the smallest unit of spatial resolution that can be represented in the database. To be more efficient, all coordinates in the database are stored as 32-bit integers in database units.

A user unit, on the other hand, is the physical unit of interest to you and is typically equivalent to many database units. The typical user unit for IC layout is one micron. The typical user unit for schematic entry is either one inch or one centimeter, depending on the standard used. Measurements made in user units are usually expressed as floating-point numbers.

Scale Factors

Within each application area, such as layout or schematics, use the same scale factor (database units per user unit) throughout your design.

- The typical scale factor used in IC layout is 100–1000 database units per micron.
- The typical scale factor used in schematic entry is 160 database units per inch (division into 1/16 of an inch) or 100 database units per centimeter.

You must make any conversions between database and user units. Because CDBA handles only database units, the scale factor used at the root of a design hierarchy applies by default to the rest of the hierarchy.

User Unit Type Attributes

Each cellview has two user unit type attributes:

- **DBUPerUU**
Stores the number of database units (DBU) per user unit (UU) that was used when the cellview was created. For example, IC layout frequently uses 1000 database units per micron.
- **userUnits**

Cadence Integrators Toolkit Database Reference

CDBA Representation of Elements

Stores the name of the unit being used as a string. The most commonly used units are metric (microns) and English (mils). The following table summarizes the choices available.

Commonly Used Units

User unit	Number per inch	Per centimeter	Per millimeter
mil	1000	393.701	39.3701
inch	1	0.393701	0.0393701
millimeter	25.4	10	1
centimeter	2.54	1	0.1
micron	25400	10000	1000



Although these values are accessible to you, NEVER use the values obtained from these functions to perform your own conversions between database units and user units.

Do ALL unit conversions using one of the many conversion functions supplied by the User Interface Manager Human Interface (HI) functions to ensure consistency in converting values back and forth. See the [Cadence User Interface SKILL Reference](#) for more information.

Most Cadence® software requires cellviews of the same type (for example, schematic) in a design hierarchy to have the same value of DBUPerUU. These attributes are normally initialized from the corresponding *sdatemplate*, and you are strongly advised against changing the value of either attribute in a cellview.

Functions for Database and User Unit Information

You can use the following functions to set and retrieve information about database and user units associated with a cellview.

dbGetCellViewDBUPerUU

double

Cadence Integrators Toolkit Database Reference

CDBA Representation of Elements

```
dbGetCellViewDBUPerUU(  
    dbCellViewId cellViewId  
);
```

Returns the database unit per user-unit attribute associated with the cellview. If the `cellViewId` does not exist, a message is printed before 0 is returned.

dbSetCellViewDBUPerUU

Boolean

```
dbSetCellViewDBUPerUU(  
    dbCellViewId cellViewId, float dbUnitsPerUserUnit  
);
```

Sets the database unit per user-unit attribute on the cellview to the given value. If the `cellViewId` does not exist, a message is printed before FALSE is returned.

dbGetCellViewUserUnitName

String

```
dbGetCellViewUserUnitName(  
    dbCellViewId cellViewId  
);
```

Returns the name of the user units used as a cellview string. If the `cellViewId` does not exist, a message is printed before `dbcNullString` is returned.

dbSetCellViewUserUnitName

Boolean

```
dbSetCellViewUserUnitName(  
    dbCellViewId cellViewId, String unitName  
);
```

Sets the name of the user units used to the given name. If the `cellViewId` does not exist, a message is printed before FALSE is returned.

Functions to Convert between Database and User Units

The following routines perform part of the conversions between database and user units. They are called from the edit-to-user-unit conversion routines.

Cadence Integrators Toolkit Database Reference

CDBA Representation of Elements

In rare circumstances in standalone applications, you might need these to transform coordinates that are not in the edit cellview. Check that you can call any of the functions documented instead of using these special functions. If the `cellViewId` does not exist or if any of the bounding boxes are not well-formed, a message is printed before returning.

dbDBToUserCoord

```
dbUserCoord
dbDBToUserCoord(
    dbCellViewId cellViewId, dbCoord coord
);
```

Converts coordinates from database units to user units.

dbDBToUserPoint

```
dbUserPoint
dbDBToUserPoint(
    dbCellViewId cellViewId, dbPoint point
);
```

Converts points from database units to user units.

dbDBToUserBBox

```
Boolean
dbDBToUserBBox(
    dbCellViewId cellViewId, dbBBox pBBox, dbUserBBox pUserBBox
); /* modifies UserBBox */
```

Converts bounding box coordinates from database units to user units. Returns FALSE if any errors are detected.

dbUserToDBCoord

```
dbCoord
dbUserToDBCoord(
    dbCellViewId cellViewId, dbUserCoord userCoord
);
```

Converts coordinates from user units to database coordinates.

dbUserToDBPoint

```
dbPoint
dbUserToDBPoint (
    dbCellViewId cellViewId, dbUserPoint userPoint
);
```

Converts points from user units to database units.

dbUserToDBBBox

```
Boolean
dbUserToDBBBox (
    dbCellViewId cellViewId, dbUserBBox pUserBBox, dbBBox pBBox
); /* modifies BBox */
```

Converts bounding box coordinates from user units to database units. Returns FALSE if any errors are detected.

Coordinates

All coordinates are in database units. A 32-bit integer represents the x or y coordinate in database units. The CDBA representation of a database coordinate is

```
typedef int dbCoord;
```

This representation indicates that `dbCoord` is defined as a 32-bit integer in CDBA, and CDBA allocates space and stores it on disk as the int data type.

Therefore, to prevent overflow and other calculation errors, restrict values to -2^{28} to $2^{28}-1$.

Points

A point is a structure containing the x and y point coordinates in a two dimensional space.

Points are so fundamental to geometrical operations that they are treated as primitive types, much like integers. Unlike other C structures used in CDBA, points are treated similarly to scalar values in function calls. Entire structures representing points are passed as function arguments and returned as function values. This greatly simplifies bookkeeping and memory allocation.

CDBA Representation of a Point

A point is represented as follows in CDBA:

```
struct dbPoint{  
    dbCoord x, y;  
};  
typedef struct dbPoint dbPoint;
```

Point Arrays

The points in a point array specify the vertices of a complex shape. Indexes for point arrays are stored internally as unsigned short (that is, 16-bit) integers. A point array can contain a maximum of 65535 points. The following are functions for point arrays:

dbCopyPointArray

```
dbPoint *  
dbCopyPointArray(  
    dbPoint pointArray, unsigned nPoints  
);
```

Allocates an array of points with size specified by `nPoints`, then copies the entire `pointArray` to the newly allocated storage. It returns a pointer to the newly created point array.

dbFreePointArray

```
Boolean  
dbFreePointArray(  
    dbPoint pointArray, unsigned nPoints  
);
```

Frees the array of points pointed to by `pointArray`, which must be allocated by `dbCopyPointArray`.

Bounding Boxes

The bounding box of a geometrical shape is defined as the smallest completely enclosed rectangle with edges parallel to the x and y axes.

The data in the box must always be well formed. The x and y coordinates of the lower-left corner must have smaller values than the x and y coordinates of the upper-right corner. For calculating the width and height of a rectangle, both the lower-left and upper-right corners are included in the rectangle. A bounding box whose lower-left corner and upper-right corner are the same contains one point.

The database automatically computes and updates the bounding boxes whenever a shape is created, modified, or deleted.

CDBA Representation of a Bounding Box

A bounding box is represented by a structure containing two points specifying the lower-left and upper-right corners of the rectangle.

```
struct dbBBox{
    dbPoint, lowerLeft, upperRight;
};
typedef struct dbBBox dbBBox;
```

Functions and Macros for Bounding Boxes

The following macros and functions retrieve information in the database about coordinates for bounding boxes and modify coordinates for bounding boxes.

The first four macros (`dbLeft`, `dbBottom`, `dbRight`, and `dbTop`) are provided for accessing the four elements in a bounding box.

dbLeft

```
dbCoord
dbLeft (
    dbBBox pBBox
);
```

Returns the x coordinate of the lower-left corner.

dbBottom

```
dbCoord
dbBottom (
    dbBBox pBBox
);
```

Returns the y coordinate of the lower-left corner.

dbRight

```
dbCoord
dbRight (
    dbBBox pBBox
);
```

Returns the x coordinate of the upper-right corner.

dbTop

```
dbCoord  
dbTop (  
    dbBBox pBBox  
);
```

Returns the y coordinate of the upper-right corner.

dbFillBBox

```
dbBBox *  
dbFillBBox (  
    dbBBox pBBox, dbCoord x1, dbCoord y1, dbCoord x2, dbCoord y2  
);
```

This macro provides a convenient template for filling in the contents of a bounding box. It lets you fill in a temporary `BBox` with x and y coordinates and pass the pointer to it as a function argument.

It fills the user-allocated bounding box `BBox` with the coordinates `x1:y1` for the lower-left corner and `x2:y2` for the upper-right corner. A pointer to the bounding box is returned for convenience. This macro is useful for filling in a temporary bounding box that can be passed as a function argument to functions such as `dbCreateRect`.

Example

```
dbBBox tempBBox;  
dbCoord x1, x2, y1, y2;  
dbCreateRect(cellViewId, layer, purpose,  
dbFillBBox(&tempBBox, x1, y1, x2, y2) )
```

dbValidBBox

```
Boolean  
dbValidBBox(  
    dbBBox pBBox  
);
```

This macro returns `TRUE` if the bounding box is well-formed, which means

- The left edge is coincident with or to the left of the right edge, and
- The bottom edge is coincident with or below the top edge

dbFixBBox

```
void
dbFixBBox (
    dbBBox pBBox
); /* modifies BBox */
```

This function swaps left and right or top and bottom of the bounding box to make a well-formed bounding box. Most functions that use bounding boxes require them to be well formed. For example, the function to compute the union of two bounding boxes does not work correctly if the left edge of the bounding box is to the right of the right edge.

dbCopyBBox

```
void
dbCopyBBox (
    dbBBox pSrcBBox, dbBBox pDstBBox
);
/* modifies DstBBox */
```

Copies a bounding box.

dbBBoxIntersect

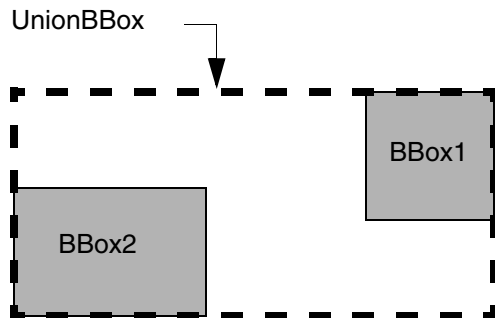
```
Boolean
dbBBoxIntersect (
    dbBBox pBBox1, dbBBox pBBox2
);
```

Determines whether two bounding boxes intersect. Returns TRUE if the two bounding boxes intersect. If the bounding boxes abut, they are considered to be intersecting.

dbUnionBBox

```
void
dbUnionBBox (
    dbBBox pUnionBBox, dbBBox pBBox1, dbBBox pBBox2
);
```

Computes the smallest bounding box that contains both `BBox1` and `BBox2`. The union bounding box is stored in `UnionBBox`.



dbUnionBBox2

```
void
dbUnionBBox2 (
    dbBBox pUnionBBox, dbBBox pBBox2
);
```

Computes the smallest bounding box that contains both `UnionBBox` and `BBox2`. The union bounding box is stored in `UnionBBox`.

dbIntersectBBox

```
void
dbIntersectBBox (
    dbBBox pIntersectBBox, dbBBox pBBox1, dbBBox pBBox2
);
```

Computes the largest bounding box that contains the common area of both `BBox1` and `BBox2`. The intersection bounding box is stored in `IntersectBBox`.

dbIntersectBBox2

```
void
dbIntersectBBox2 (
    dbBBox pIntersectBBox, dbBBox pBBox2
);
```

Computes the largest bounding box that contains the common area of both `IntersectBBox` and `BBox2`. The intersection bounding box is stored in `IntersectBBox`.

User-Defined Coordinates for Bounding Boxes

The data structures for user-defined coordinates for bounding boxes are similar to the corresponding data structures that store database coordinates for bounding boxes as shown in [“CDBA Representation of a Bounding Box”](#) on page 135.

```
typedef double dbUserCoord;
struct dbUserPoint {
    dbUserCoord      x, y;
};
typedef struct dbUserPoint dbUserPoint;
struct dbUserBBox {
    dbUserPoint      LowerLeft, upperRight;
};
typedef struct dbUserBBox dbUserBBox;
```

Functions and Macros for User Coordinates

The following functions and macros retrieve user-defined information about coordinates of bounding boxes and modify coordinates of bounding boxes.

dbUserLeft

```
dbUserCoord
dbUserLeft(
    dbUserBBox pUserBBox
);
```

Returns the x coordinate of the left edge of the bounding box.

dbUserBottom

```
dbUserCoord
dbUserBottom(
    dbUserBBox pUserBBox
);
```

Returns the y coordinate of the bottom edge of the bounding box.

dbUserRight

```
dbUserCoord
dbUserRight(
    dbUserBBox pUserBBox
);
```

Returns the x coordinate of the right edge of the bounding box.

dbUserTop

```
dbUserCoord  
dbUserTop(  
    dbUserBBox pUserBBox  
);
```

Returns the y coordinate of the top edge of the bounding box.

dbFillUserBBox

```
dbUserBBox *  
dbFillUserBBox(  
    dbUserBBox pUserBBox, dbUserCoord x1, dbUserCoord y1, dbUserCoord x2,  
    dbUserCoord y2  
);
```

This macro provides a convenient template for filling in the contents of a bounding box. It fills the user-allocated bounding box, `UserBBox`, with the coordinates `x1:y1` for the lower-left corner and `x2:y2` for the upper-right corner. If necessary, it swaps coordinates to make a well-formed bounding box. A pointer to the bounding box is returned for convenience. This macro is useful for filling in a temporary bounding box so it can be passed as an argument to various functions.

dbFixUserBBox

```
void  
dbFixUserBBox(  
    dbUserBBox pUserBBox  
); /* modifies UserBBox */
```

This function swaps the coordinates, if necessary, to make the data within the box well-formed (with the lower-left corner below and to the left of the upper-right corner).

Angles

All angles in CDBA are specified in radians and measured counterclockwise from the x axis. Radians are used, rather than the more familiar degrees, to be consistent with the trigonometric functions defined in the mathematical library of the C language.

Because of the limited precision inherent in floating-point arithmetic, never test for strict equality between angles, which are represented as floating-point numbers. Use appropriate “fudge” factors when making comparisons between floating-point numbers.

CDBA Representation of an Angle

The following is a type definition for angle (in radians):

```
typedef double dbAngle;
```

Functions to Convert between Degrees and Radians

For convenience, two macros are provided to convert between degrees and radians:

dbDegreeToRadian

```
dbAngle  
dbDegreeToRadian(  
    double angleInDegrees  
);
```

Converts the measurement of an angle from degrees to radians.

dbRadianToDegree

```
double  
dbRadianToDegree(  
    dbAngle angleInRadians  
);
```

Converts the measurement of an angle from radians to degrees.

Shape Transformation

To transform shapes, use an orientation (angle and mirroring information), an offset in the x and y directions, and a magnification. This information is encapsulated in the `dbTransform` structure, which is treated like a scalar in argument passing.

CDBA Representation of Shape Transformation

```
struct dbTransform{ dbOrient orient, dbPoint offset, float mag, };
```

```
typedef struct dbTransform dbTransform;
```

The constant `dbcIdentTransform` represents the identity transformation. The variable `dbvIdentTransform` is a variable initialized to the identity transformation.

Functions to Transform Shapes

Use the following functions to transform shapes and perform additional operations on the transformations.

dbInvertTransform

```
dbTransform  
dbInvertTransform(  
    dbTransform transform  
);
```

Inverts the transformation on `transform`, and returns the inverted transformation as the value of the function.

dbConcatTransform

```
dbTransform  
dbConcatTransform(  
    dbTransform trans1, dbTransform trans2  
);
```

Concatenates the second transformation onto the first transformation and returns it as the value of the function. This process is a matrix multiplication that applies `transform1` and then `transform2`. A message is printed if arithmetic overflow is detected.

dbTransformPoint

```
dbPoint  
dbTransformPoint(  
    dbPoint point, dbTransform transform  
);
```

Transforms the point passed in and returns the new point. A message is printed if there is arithmetic overflow.

dbTransformPointArray

```
void
dbTransformPointArray(
    dbTransform transform, dbPoint pBefore, dbPoint pAfter, int nPoints
);
```

Transforms the point array passed in and returns the new point array referred to by `pAfter`. The argument `nPoints` specifies the number of points in the array.

dbTransformBBox

```
Boolean
dbTransformBBox(
    dbBBox pOldBBox, dbTransform transform, dbBBox pNewBBox
); /* modifies NewBBox */
```

Transforms a bounding box. Always returns TRUE.

dbTransformFigRec

```
Boolean
dbTransformFigRec(
    dbFigRec pFigRec, dbTransform transform
);
```

Transforms the figure record pointed by `pFigRec`. Returns FALSE if the transformation of an unsupported figure type is attempted.

`dbTransformAccessDirection` is documented in [“Pins”](#) on page 746.

Distance, Direction, and Time

The `dbDistance` type, which is declared as an unsigned int, measures scalar distances between objects in the database.

The `dbDirection` enumeration describes the direction to search for an object from a reference bounding box. A `dbDirection` can take one of the values `dbcTop`, `dbcBottom`, `dbcLeft`, or `dbcRight`. `dbDirection` is used by various neighbor routines, such as `dbGetNeighbor` and `dbGenNeighbor`.

The `dbTime` type is declared as `time_t`, which is the C language type representing time.

dbConvertTimeString

dbTime

```
dbConvertTimeString(  
    String timeString  
);
```

Converts `timeString` to a `dbTime` type variable.

dbGetCreateTime

time_t

```
dbGetCreateTime( dbCellViewId cellViewId  
    );
```

Returns the time at which the design was created.

CDBA Design Management Functions

The following is required to manage a design:

- [Design Data Management System](#) on page 146
- [Design Library](#) on page 146
- [Library List](#) on page 148
- [Library List Generator Functions](#) on page 148
- [Data Organization](#) on page 150
- [Object Information Retrieval Functions](#) on page 150
- [File Location for Reading and Writing](#) on page 159
- [Synchronization and Access Protection](#) on page 159
- [File Checkout and Checkin](#) on page 160

Design Data Management System

The design data management system has the following objectives:

- Access a design file through logical names
- Provide version control over design data
- Define and reassign a new working context
- Provide hooks for application programs
- Keep track of design evolutions

Some of the functions that appear to be provided by CDBA system calls are provided by a data management system called the Design Data Procedural Interface (DDPI). The DDPI handles the low-level manipulation of the libraries, cells, and views. The DDPI also presents a common programming interface for calling the `cdmcheckin` and `cdmcheckout` programs, which can handle multiple design management systems.

The procedural interface (PI) is made up of sets of functions. Many of these functions need to handle objects differently, depending on the object's type and location. To facilitate these differing needs, whenever a PI function locates an object, it returns a `ddId`, which other PI functions can use to identify the type and location of the corresponding object.

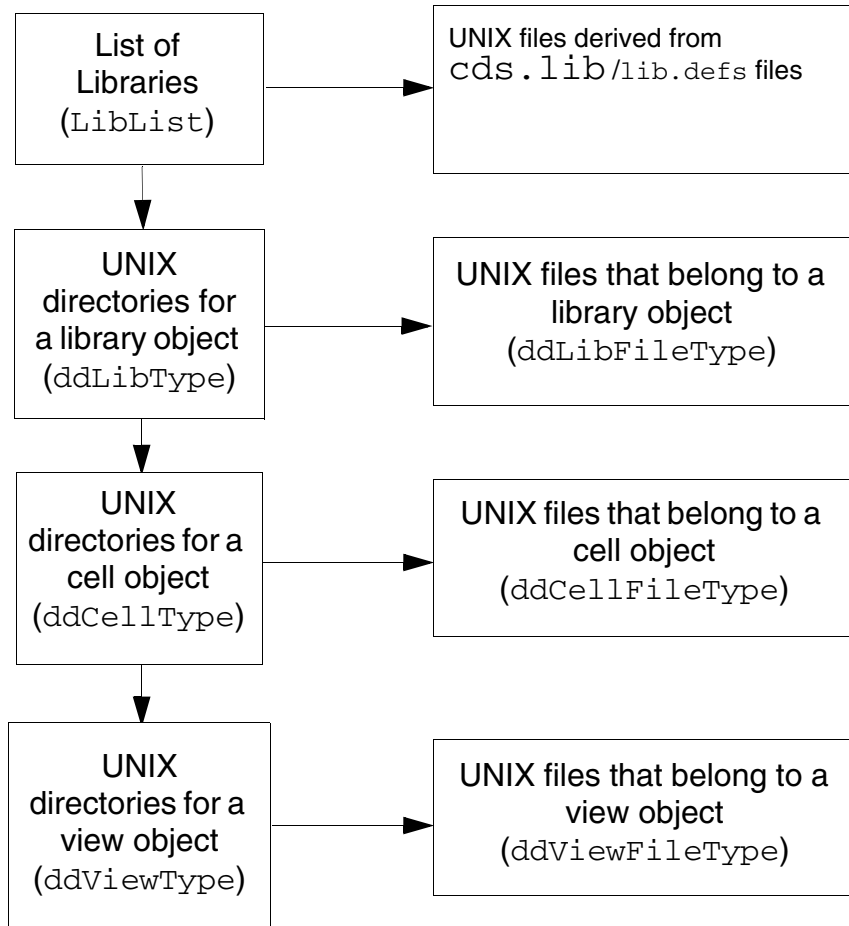
Note: `ddIds` are unique within the lifetime of the process in which they are issued and are local to that process. They must not be stored in files or databases and are not valid after the process dies.

This chapter describes the library procedures available under the DDPI. Higher-level tools can be built on top of these procedures.

Design Library

A design library is a collection of cells, cellviews, and data files under a particular cell and cellview, and of different versions of the data files. It provides a logical view of the design data for the design data management system. The data file locations are transparent to the user. For information about

A design library is organized into four levels. The following illustration shows these levels, and the types of directories and files they contain.



Startup File Locator

The following function returns the path to the startup file (such as the `cds.lib` file) you specify. The `contextId` should be NULL for ITKDB users.

```
String  
ddGetStartup(  
    String fileName, ddId contextId  
);
```

Specify the search path for startup files in the `setup.loc` file, in either

```
$CDS_SITE/cdssetup/setup.loc
```

or

`your_install_dir/share/cdssetup/setup.loc`

Because the path returned is a pointer to a static character buffer, copy this path before you make additional calls to `ddGetStartup`.

Library List

Tools locate the files they need to access by looking in several standard places. Most tools have data search-path functions that enable users to look at a list of places. A library list indicates the search paths for files.

Functions to Generate the Library List

The DDPI supplies the functions `ddStartGenLibList`, `ddGenLibList`, and `ddStopGen` to generate and examine the library list.

Function to Identify a 5x Library

The DDPI supplies the function `ddLibIs5X` to determine whether the library was created as a 5X library.

`cds.lib` and `lib.defs` Files

For information about `lib.defs` keyword and syntax, DDPI and library manager support for `lib.defs` and `cds.lib` files, see [Cadence Library Definition File](#), in Chapter 5 of the *Cadence Application Infrastructure User Guide*.

Currently, ITKDB does not allow its users to dynamically edit the `cds.lib` or `lib.defs` when running their ITKDB-based tools. Also, it does not allow its users to use the `chdir` function in their code.

Library List Generator Functions

You can use a library list generator to obtain the list of libraries in virtual memory (VM). The following generator functions work together to find the libraries in a library list.

ddStartGenLibListLib

```
ddLibListGenState *  
ddStartGenLibListLib(  
    );
```

Initializes a library list (`LibList`) library generator.

ddGenLibListLib

```
ddId  
ddGenLibListLib(  
    ddLibListGenState *pState  
    );
```

Finds the next library in the library list given a library list generator. Returns NULL when it finishes.

ddStopGen

```
void  
ddStopGen(  
    voidstar a  
    );
```

Releases the space occupied by the generator pointed to by the parameter `a`. Function to identify a 5X library

Function to Identify a 5X Library

The following function determines whether a library is a 5X compatible library.

ddLibIs5X

```
Boolean  
ddLibIs5X(  
    ddId libId  
    );
```

OpenAccess supports different types of design management systems using plug-ins. DFII on OpenAccess supports only the FileSys plug-in which provides backwards compatibility with the 5X library structure. This function determines whether a library was created with the 5X library structure.

Returns TRUE if the library was created as a 5X library. Returns False otherwise.

Data Organization

Many tools have the following model for organizing data:

- A library contains collections of module-specific data called cells.
- Cells contain design-specific collections of data called views.
- Views contain files (cellviews).

The following table shows the design data (DD) ID types of the objects that correspond to the libraries, cells, views, and the files contained in them.

Object	DD ID type
Library	ddLibId
File belonging to a library	ddLibFileId
Cell	ddCellId
File belonging to a cell	ddCellFileId
View	ddViewId
File belonging to a view	ddViewFileId

Object Information Retrieval Functions

The following functions retrieve information about objects.

ddIsId

```
Boolean  
ddIsId(  
    ddId id  
);
```

Returns TRUE if the parameter `id` is a 32-bit opaque ID known to the PI, and returns FALSE otherwise.

ddGetObjName

```
String  
ddGetObjName (  
    ddId obj  
);
```

Returns the name of the object.

ddDeleteObj

```
Boolean  
ddDeleteObj (  
    ddId obj  
);
```

Deletes the object and removes it from use. Depending on the design management integration, the object may simply be removed from the next release of the project yet remain in the repository, or the object may be completely removed along with its history.

If the underlying design management system is `designSync`, the design manager removes the object from the next release of the project, but the object remains in the repository. However, if the underlying design management system is `versionSync`, the design manager deletes the object from the repository, as both the file and its entire history are removed.

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 detects that another process is using that file. It therefore cannot delete the object, and it returns `FALSE` without deleting anything.

If `obj` 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 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 `obj` is of any other type, `ddDeleteObj` removes the object from disk from the master directory.

ddGetObj

```
ddId  
ddGetObj (  
    String libName,  
    String cellName,  
    String viewName,  
    String fileName,  
    ddId contextId,  
    String mode  
);
```

Description

Finds libraries, cells, views, and files, and creates cells, views, and files. The simplest usage is to find a library by name:

```
library = ddGetObj("libName", NULL, NULL, NULL, NULL, "r")
```

Note: If the file you want is a cellview, use `dbOpenCellViewByType`. Do not use `ddGetObj`. The `dbOpenCellViewByType` function calls `ddGetObj` for you.

DDPI uses the following concepts to define the behavior of `ddGetObj`:

- A master file is a file that the writing application looks to for source data or data that cannot be recreated without user input. There can be only one master file per view. The master file might be identified by the file belonging to the view named `master.tag` containing the name of the master file.
- A co-master file is a file that the writing application looks to for source data or data that cannot be recreated without user input but is useless without the master. Examples include 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 the `contextId` for the master. If writing or appending to the file, use the `d` modifier in the `mode` parameter.

- The `relative name` for a file is specified relative to a library. Files can belong to the library, cell, or view. The name consists of

```
[/cellName[/viewName]]/fileName
```

The `ddGetObj` function searches for the object under your specified library in the `cds.lib` files until it finds the first occurrence of the “relative named” object. If you

Cadence Integrators Toolkit Database Reference

CDBA Design Management Functions

requested read access, the function is done and returns an `id`. It returns an error if the tool does not have read access to that object.

If you request any other access mode, `ddGetObj` first searches for the object as it does in read mode, then returns a value according to whether or not the file exists.

`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>libName</code>	Specifies the library name to find the object you want. The named library must appear in the <code>LibList</code> . If you do not specify a <code>libName</code> but you specify a <code>contextId</code> , DDPI takes the <code>libName</code> from the context object.
<code>cellName</code>	Specifies the name of a cell in the path to the object you are looking for. <code>ddGetObj</code> looks for the cell in the library named by <code>libName</code> . If you do not specify a <code>cellName</code> but you specify a <code>contextId</code> , DDPI takes the <code>cellName</code> from the context object.
<code>viewName</code>	Specifies the name of a view in the path to the object you are looking for. <code>ddGetObj</code> looks for the view in the library specified by <code>libName</code> , in the cell specified by <code>cellName</code> . If you do not specify a <code>viewName</code> but you do specify a <code>contextId</code> , then DDPI takes the <code>viewName</code> from the context object.

Switch-List Behavior

The `viewName` parameter can (optionally) 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 `viewNames` from the list. DDPI returns successfully if it finds a match with any of the `viewNames`. 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.

Cadence Integrators Toolkit Database Reference

CDBA Design Management Functions

`fileName` Indicates you are looking for a file object and specifies its name. `ddGetObj` looks for the file in the named library only.

Wildcard Behavior

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 `viewFiles`. 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.

`contextId` You can specify a `contextId` object from which to take the library, cell, or view names. This parameter provides a way to find an object that is in the same directory or has the same parents as some other object.

mode

Controls side effects of `ddGetObj`.

The following values are allowed:

- `r`
Gets the ID of an existing object in the specified library.
- `w`
Gets the ID for the 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 fails if a master object already exists in the specified view.
- `wc`
This option is for creating co-master data and is similar to `w` except that if the object is a viewfile, it is created without marking it as a master object. This succeeds even if there is a master object in the view.
- `wd`
This option is for creating derived data and is similar to `w` except that the object is created only if the object specified by `b_contextId` is a master. This file is co-located with the master object.
- `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.
- `ad`
Gets the ID of an existing derived object in the specified library. You can open it for modification. The object will be created if it does not exist and behaves the same as `wd`.

The `ddGetObj` function might create new cells, views, and files as a side effect to `w`, `wd`, `a`, or `ad` mode requests when the object is created.

Here are object type parameters to specify:

- `libName` to create or access a library.
- `libName` and `fileName` to create or access a library file.
- `libName` and `cellName` to create or access a cell.
- `libName`, `cellName`,* and `fileName` to create or access a cell file.
- `libName`, `cellName`,* and `viewName` to create or access a view.
- `libName`, `cellName`,* `viewName`,* and `fileName` to create or access a view file.

Important

If a `contextId` is specified, the fields with an asterisk (*) do not need to be specified, and the `libName` must not be specified.

ddGetObjLib

```
ddId
ddGetObjLib (
    ddId obj
);
```

Finds the ID of the library path object that contains it when given any `ddId`. If the object is a library, it returns the same ID.

ddGetObjType

```
ddType
ddGetObjType (
    ddId obj
);
```

Returns the `ddType` of the object when given the `ddId` of an object.

ddGetTypeNames

```
String  
ddGetTypeNames(  
    ddType type  
);
```

Returns a string that contains the name of the value, when given an `enum` of type `ddType`. This name must be copied before the next call to `ddGetTypeNames` because DDPI allocates it in a static buffer.

ddGetMasterSet

```
ddGetMasterSet(  
    g_viewId  
);
```

Description

Returns the list of files in the Master Set (if defined) for the given `viewId` or returns `nil`. The Master Set is a subset of the Co-Managed Set composed of master data.

Arguments

<i>g_viewId</i>	A <i>viewId</i> returned by <code>ddGetObj</code> .

Value Returned

<i>l_masterSetList</i>	The list of files in the Master Set for the given <code>viewId</code> .
<i>nil</i>	The Master Set is not defined.

Example

```
viewId=ddGetObj("mylib" "mycell" "layout" "layout.oa" nil "w")  
masterSet=ddGetMasterSet(viewId)
```

returns

```
("layout.oa" "master.tag")
```

ddStartGenObjRel

```
ddObjRelState*
ddStartGenObjRel (
    ddId obj, ddObjRelType relType
);
```

Initializes a generator to get the list of

- Cells or files belonging to a library
- Views or files belonging to a cell
- Files belonging to a view

The `obj` parameter specifies the parent object, and the `relType` parameter specifies whether you want the list of child objects or of files. Returns NULL if there are no objects with the specified relationship.

If an object gets created while the generator is active, it is not made visible until the next call to `ddStartGenObjRel`. If an object is deleted after the generator is started but before you access the object, you might get an “object does not exist” warning when you access it. The `ddStartGenObjRel` function does not get you any object locks.

ddGenObjRel

```
ddId
ddGenObjRel (
    ddObjRelState* pGen
);
```

Returns the next entry in the list of files each time you call `ddGenObjRel` in the generator state specified by `pGen`. When the list is exhausted, it returns NULL.

ddStopGen

```
void
ddStopGen (
    voidstar a
);
```

Releases the space occupied by the generator pointed to by the parameter `a`. This function works with all DDPI generators.

File Location for Reading and Writing

The following functions enable a tool to read from or write to files when it needs to.

ddGetObjReadPath

```
String  
ddGetObjReadPath(  
    ddId obj  
);
```

Returns the read path to the object when given the `ddId` for the `obj`. Because the path that DDPI returns is not persistent, copy it before making any other `dd` or `db` calls.

ddGetObjWritePath

```
String  
ddGetObjWritePath(  
    ddId, obj  
);
```

Returns the write path to the object when given the `ddId` for `obj`. Because the path that DDPI returns is not persistent, copy it before making any other `dd` or `db` calls.

Synchronization and Access Protection

As workbenches become collections of cooperating tools, the need increases for tools to synchronize their access to files. As Cadence® design framework II design manager goes away, DDPI provides similar file-access synchronization services using this procedural interface.

The following two mechanisms are needed to support cooperative work:

- Access protection so only appropriate files can be accessed (using UNIX mechanisms)
- Long transactions based on `checkin` and `checkout`

Multiple users concurrently employing tools need long transactions supplied by a design management system. Checkin and checkout behavior is optional. The procedural interface does nothing unless the data is managed by a design management system.

File Checkout and Checkin

The following functions work together to check files out and back in.

ddNeedCheckout

```
Boolean  
ddNeedCheckout(  
    ddId file  
);
```

Returns TRUE if the file (given a `ddId` for a file object) is a DM-managed object that needs to be checked out. Returns FALSE if the library is not DM managed or if the file does not need to be checked out.

`ddNeedCheckout` exists only so tools that need to provide automatic checkout behavior can determine if they need to call the `ddCheckout` and `ddCheckin` functions. An example follows:

```
/* Suppose you want to update a file that belongs to a library */  
ddId fileId = ddGetObj(NULL, NULL, NULL, fileName, libId, "w");  
  
/* NOTE ddGetObj() only returned the id, it didnot open anything  
* so it didnot find that the file was not writable.  
*/  
if(ddNeedCheckout(fileId)){ /* See if it needs checkout */  
    ddId fileList[2];  
    fileList[0] = fileId;  
    fileList[1] = 0;  
    void) ddCheckout(fileList,""); /* Check it out */  
}
```

ddCheckout

```
Boolean  
ddCheckout(  
    ddId* files, String switches  
);
```

Builds a system command that calls the `cdmcheckout` command. The `cdmcheckout` command makes it possible to integrate Cadence tools with Revision Control System (RCS) and other version control applications. The `files` parameter is a NULL-terminated array of `ddIds`.

Cadence Integrators Toolkit Database Reference

CDBA Design Management Functions

The switches allowable are `r` and `v`

- `r` Checks out files as read-only.
- `v` Specifies a particular version of a file to be checked out. You must always follow this switch with the version of the file you are checking out.

ddCheckin

Boolean

```
ddCheckin(  
    ddId* files, String description, String switches  
);
```

Builds a system command that invokes the `cdmcheckin` command. `cdmcheckin` makes it possible to integrate Cadence tools with Revision Control System (RCS) and other version control applications. The `files` parameter is a NULL-terminated array of `ddIds`. The switches allowable are `c` and `f`

- `c` Cancels the `checkout` of the named files.
- `f` Forces the `checkin`, even if the file did not change.

Cadence Integrators Toolkit Database Reference

CDBA Design Management Functions

CDBA Input and Output Functions

This chapter covers the following topics on input and output functions:

- [UNIX Directory Paths](#) on page 164
- [Routines Returning Directory Paths](#) on page 164
- [Initialization and Exit Functions](#) on page 165
- [Input and Output Functions](#) on page 167
- [Hierarchy Traversal Functions](#) on page 182
- [Operations on Cellviews](#) on page 183
- [Counter Functions](#) on page 197
- [Database I/O Functions](#) on page 204

UNIX Directory Paths

The UNIX file system is based on a hierarchical directory structure. The base of the directory hierarchy is known as the root and is represented by the slash character (/) as the first character in a directory or file name. In UNIX terminology, the traversal from the root directory, through descendent directories, to a particular file in the hierarchy, is called a path. Thus, a directory path is equivalent to a UNIX filename.

In the UNIX environment, there are two ways to refer to a file name: “full directory path” and “relative directory path.” The full directory path specifies the means of traversal from the root directory to the target file. An example is

```
/cds/etc/cdslib/schema/nand2/schematic/sch.oa
```

The relative directory path does not begin with a slash (/). For example, if /usr/me/mylib is the current directory, the directory under which you started the program, you can also use the directory path `nand2/schematic/sch.oa` to refer to the following file:

```
/usr/me/mylib/nand2/schematic/sch.oa
```

CDBA recognizes the following directory naming conventions:

- . Refers to the current directory
- .. Refers to the directory above the current directory
- ~ As understood by the C and Korn shells (`cs`h and `ks`h)

Routines Returning Directory Paths

Certain routines return a UNIX directory path, or file name, to the calling routine. For example, `dbGetCellViewFullDirPath(cellViewId)` returns the full directory path for the file containing the database representation corresponding to `cellViewId`. The string returned is called a cellview string, because it is contained in the cellview in virtual memory. When the cellview is closed, the cellview string becomes invalid.

Initialization and Exit Functions

The following functions initialize and exit CDBA.

dbInit

```
Boolean  
dbInit(  
    int argc, char argv  
);
```

Description

Starts all applications. You must call the `dbInit` function in the first executable statement of the `main` routine for the applications to initialize the CDBA functions. `dbInit` might fail if your system does not have adequate virtual memory or has not been properly configured.

The `argc` and `argv` variables are the same variables used by the C language to pass command line arguments to the main program. Notice the first argument to `dbInit` is a pointer to `argc` rather than the `argc` variable itself. This is because `dbInit` scans the command line arguments searching for special switches intended for CDBA.

If such switches are found, `dbInit` modifies the command line arguments, stored in `argc` and `argv`, to remove the switches intended for CDBA. All command line switches used by CDBA begin with the prefix `db` and follow the standard CDBA naming conventions.

dbExit

```
Boolean  
dbExit(  
    );
```

Description

Exits CDBA. You call `dbExit` after all database work is complete (usually just before calling `exit`). The `dbExit` function releases all libraries locked when CDBA is run, and purges any libraries opened within CDBA. The `dbExit` function always returns TRUE.



Do not call any database functions after calling `dbExit`.

ilInitClients

```
void
ilInitClients(
    int *, char *[], void *, int
);
```

Description

Initializes SKILL within the itkDB environment. This function is called after the `dbInit()` call. Without a call to this function:

- SKILL PCells are not evaluated within itkDB.
- SKILL startup files, such as `libInit.il` associated with techLibs are not loaded properly.

Additionally, to use this function for itkDB applications, set the third argument to `NULL` and the fourth argument to `0`.

Example

```
if ( !dbInit(&argc, argv) )
{
    (void) fprintf(stderr, "dbInit failed.\n");
    exit( 1 );
}

ilInitClients(&argc, argv, NULL, 0); /*initializes SKILL within itkDB environment
*/
```

Input and Output Functions

The following are input and output functions for libraries and cellviews.

Libraries

The following functions retrieve information about libraries.

dbCreateLib

```
ddId  
dbCreateLib(  
    String libName, String libPath  
);
```

Given the name of a library and a path to a library component directory, the system searches for the named library in the `cds.lib`. If the named library exists, DDPI (Design Data Procedural Interface) returns the ID of the library. If the named library does not exist, DDPI creates the corresponding directory and adds the `DEFINE` command to the nearest library list file (such as the one the DDPI found first searching with `ddUpdateLibList`). If you call `dbCreateLib` and there is no visible `cds.lib` file, DDPI creates one in the current working directory (`/cwd`).

If you specify a NULL library path, then DDPI uses `cwd` and the `libName` to make a path (`cwd/libName`). If you specify a leaf name (for example, without embedded separators), DDPI makes a path out of that name and `cwd` (`cwd/libPath`). If you specify a period (`.`), DDPI does the same as for NULL.

`dbCreateLib` calls back trigger functions registered on the hooks: `PreCreateLib` and `PostCreateLib`.

dbGetLibName

```
String  
dbGetLibName(  
    ddId libId  
);
```

Returns the name of the specified library.

Cellviews

The following functions retrieve information about and act on cellviews.

dbOpenCellViewByType

```
dbCellViewId  
dbOpenCellViewByType(  
    String libName, String cellName, String viewList, String viewTypeName, String  
    mode, dbId contextId  
);
```

Description

Opens the cellview in the read, append, or write mode. For the newly created cellview, the `cellViewType` is set by mapping the `viewTypeName` to `dbCellViewType`, which is an enumerator. If the mapping fails, the `cellViewType` is set to `stranger`.

If the cellview is opened in the read mode, it must already exist. If the cellview is opened in the other modes, it is created if it does not exist.

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

Arguments

<code>libName</code>	Specifies the library name. You can enter the <code>libName</code> as <code>nil</code> (either <code>NULL</code> or a blank space), in which case the <code>contextId</code> must be given, and the search is done on the library containing the <code>contextId</code> .
<code>cellName</code>	Specifies the cell name. The <code>cellName</code> cannot contain a forward slash (/) in a cell name. Thus, the paths <code>@/cellName</code> and <code>parentCellName/childCellName</code> are not supported.
<code>viewList</code>	Specifies a blank-separated list of view names such as <code>viewA viewB</code> . The function <code>dbOpenCellViewByType</code> opens the cellview whose view name matches one on the list. If there is no match and the mode is <code>a</code> , <code>w</code> , <code>s</code> , <code>ac</code> , <code>ad</code> , <code>wc</code> , <code>wd</code> , <code>sc</code> , or <code>sd</code> , a new cellview is created from the first view in the <code>viewList</code> .
<code>viewTypeName</code>	Specifies the <code>viewType</code> , which is mapped to the <code>fileName</code> that contains the cellview data. The mapping is done by mapping the <code>viewTypeName</code> in a registry file, which must be accessible by <code>ddGetStartup</code> . If <code>viewTypeName</code> is entered as <code>*.oa</code> , <code>NULL</code> , or a blank space, no mapping is needed, but the mode must be <code>r</code> . That is, the cellview must exist. In this case, it depends on what <code>ddGetObj</code> finds when it passes in <code>*.oa</code> as the <code>fileName</code> argument. No cellview is created if the mapping fails.

The following is a list of valid `viewTypeNames`:

- `netlist`
- `schematic`
- `schematicSymbol`
- `maskLayout`

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

mode

Specifies the access mode to the cellview. The mode can be one of the following values:

- `r`
Opens the cellview in read mode. The `cellview` must already exist.
- `a`
Opens the existing cellview in append mode. If the cellview does not exist, it is created.
- `w`
Opens the cellview in write mode. If the cellview does not exist, it is created. If the cellview already exists, its contents are truncated in virtual memory.
- `s`
Opens the cellview in scratch mode. If the cellview does not exist, it is created. The cellview cannot be saved.
- `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/view.
- `wd`
Same as `wc`, except that the context cellview must be specified. `LibName` is ignored if specified. The library is derived from the context cellview.
- `ac`
Same as `a` if the cellview already exists. If the cellview does not exist, this mode is the same as `wc` mode.
- `ad`
Same as `a` if the cellview already exists. If the cellview does not exist, this mode is the same as `wd` mode.
- `sc`
Same as `s` if the cellview already exists. If the cellview does not exist, this mode is the same as `wc` mode, except the cellview cannot be saved to disk.
- `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.

- `contextId`
Specifies the cellview ID of the context cellview. This `contextId` is also used to search for the cellview in the library, as described below:
 - If only the `libName` is specified, the named library is searched.
 - If only the `contextId` is specified, the library of the context cellview is searched.
 - If both `libName` and `contextId` are specified, the library of the context cellview is searched. The argument `libName` is ignored.

If both `libName` and `contextId` are specified, the library of the context cellview is searched. The argument `libName` is ignored.

dbReOpen

```
dbCellViewId  
dbReOpen (  
    dbCellViewId cellViewId, String modeString  
);
```

Changes the input/output mode of the cellview specified by `cellViewId`. `dbOpen` already brought `cellViewId` into virtual memory to the mode specified by `modeString`. Reference count is not changed. `dbReOpen` returns NULL if it cannot open the cellview.

Note: Cellview is under access control. The cellview can be reopened if and only if the following conditions are met:

- `dbReOpen` opens the library in a mode that lets the cellview be reopened in the mode specified by `modeString`.
- The user has enough access privileges to reopen the cellview in the specified mode.
- The cellview can be reopened without violating concurrency control rules.

`dbReOpen` automatically checks out the designated cellview from the library if the requested mode is either write or append, and the designated cellview is not checked out. The user must check in the cellview after modification is completed and verified.

Note: If the `modeString` is in read mode, and the cellview has been modified and not saved, `dbReOpen` reopens the cellview without any warning.

dbOpenDerivedCellView

```
dbCellViewId  
dbOpenDerivedCellView(  
    dbCellViewId cellViewId, String viewName  
);
```

Opens a new view for the same cell with **w** (write) mode. `dbOpenDerivedCellView` calls `dmMakeViewClone` to ensure that the new view exists and has the proper type.

dbClose

```
Boolean  
dbClose(  
    dbCellViewId cellViewId  
);
```

Closes the cellview and decrements the reference count by one. If the reference count is zero, CDBA might automatically purge the cellview from virtual memory. (CDBA can cache frequently used cellviews in virtual memory to minimize disk I/O.) To force removing a cellview from virtual memory, use `dbPurge`.

dbFindOpenCellView

```
dbCellViewId  
dbFindOpenCellView(  
    dbId libId, String cellName, String viewName, String versionName  
);
```

Returns the cellview of the specified cell, view, and version name, if found and opened. Otherwise, it returns NULL. If `versionName` is a NULL string, `dbFindOpenCellView` uses the default version defined by the current configuration of the library.

dbGetCellView

```
dbCellViewId  
dbGetCellView(  
    dbId id  
);
```

Returns the cellview containing the database object identified by `id`.

dbPurge

Boolean

```
dbPurge(  
    dbCellViewId cellViewId  
);
```

Purges the cellview from virtual memory, regardless of the reference count. This function is useful because it might be impossible to match the number of times `dbClose` and `dbOpen` are called.

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`.

dbCheck

Boolean

```
dbCheck(  
    dbCellViewId cellViewId  
);
```

Checks the integrity of the database. If you specify a NULL cellview ID, then all opened cellviews are checked. The function returns TRUE if the database is OK. Otherwise, it prints warnings and returns FALSE. If you are experiencing corrupted databases, you can call this function to help determine where the corruption is occurring.

dbSave

Boolean

```
dbSave(  
    dbCellViewId cellViewId  
);
```

Saves the given cellview on disk if the cellview was modified but not saved.

Note: When you use `dbSave`, undo checkpoints for all edit actions before the save operation are cleared. To avoid losing the undo checkpoint information while saving a cellview, set the `dbUndoAcrossSave` cdsenv variable value to `t`.

dbSaveAll

Boolean

```
dbSaveAll(  
);
```

Saves all cellviews that were modified but not saved on disk in virtual memory.

dbRegSaveTrigger

Boolean

```
dbRegSaveTrigger(  
    dbCVTriggerFunc    triggerFunc,  
    unsigned int        priority,  
    const char          *name  
);
```

Registers a 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.

dbUnregSaveTrigger

Boolean

```
dbUnregSaveTrigger(  
    dbCVTriggerFunc    triggerFunc  
);
```

Un-registers the pre-save trigger function.

dbSearchCDF

Boolean

```
dbSearchCDF(  
);
```

Sets and un-sets a flag that will cause CDF to be searched for parameter values. t sets, nil un-sets the flag. There is no return value.

dbStartGenCellView

dbGenStateId

```
dbStartGenCellView(  
);
```

Starts the generator for all cellviews currently open and in virtual memory.
`dbStartGenCellView` returns NULL if there are no cellviews in virtual memory.

dbGenCellView

Boolean

```
dbGenCellView(  
    dbGenStateId pState, dbCellViewId pCellViewId  
); /* modifies CellViewId */
```

Generates all cellviews currently open and in virtual memory. dbStartGenCellView returns NULL if there are no cellviews in virtual memory.

dbGetLibDataCompressionLevel

unsigned int

```
dbGetLibDataCompressionLevel(  
    ddId  
);
```

Returns the data compression level set on the specified library ID.

dbSetLibDataCompressionLevel

Boolean

```
dbSetLibDataCompressionLevel(  
    ddId,  
    unsigned int  
);
```

Sets the data compression level for designs that are saved to the specified library ID.

A compression level indicates the amount of effort the compression algorithm uses when compressing data. A higher level does not necessarily correspond to better compression efficiency. The supported levels range from 1 to 9. Level 1 provides an optimized balance between efficiency and performance.

Note: Changing the compression level of the library only affects the designs that are saved to the library after the compression level is set. It does not recompress the existing designs in the library.

dbUnsetLibDataCompressionLevel

Boolean

```
dbUnsetLibDataCompressionLevel(  
    ddId,  
);
```

Removes the data compression level of the specified library.

dbRefreshLibDataCompressionLevel

Boolean

```
dbRefreshLibDataCompressionLevel(  
    ddId,  
    );
```

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.

dbIsNonSKILLPcell

Boolean

```
dbIsNonSKILLPcell(  
    dbCellViewId    cellViewId  
    );
```

Description

Checks whether a Pcell is not a SKILL Pcell.

Arguments

<i>cellViewId</i>	Database ID for a cellview.

Return Value

Boolean	Returns TRUE if the Pcell is not a SKILL Pcell. Returns FALSE if the Pcell is a SKILL Pcell.

dbErrorFlushWarn

```
void  
dbErrorFlushWarn(  
    void  
);
```

Description

Ensures that all warnings that were produced by database functions are written to the output.

Arguments

None.

Return Values

None.

Example

```
dbCellViewId cvId = dbOpenCellViewByType(argv[1], argv[2], argv[3], NULL,  
(char*)"r", (dbCellViewId)NULL);  
  
if (!cvId) {  
    dbErrorFlushWarn();  
    return false;  
}
```

dbHasAutoSavedFile

Boolean

```
dbHasAutoSavedFile(  
    const char    *libName,  
    const char    *cellName,  
    const char    *viewName  
);
```

Description

Determines whether the specified cellview has an auto-saved file.

Arguments

<i>libName</i>	Name of a library.
<i>cellName</i>	Name of a cellview.
<i>viewName</i>	Name of a view.

Return Value

<i>Boolean</i>	TRUE if the auto-save file is present in the specified cellview. FALSE if the auto-save file is not present in the specified cellview.

Example

```
Boolean result = dbHasAutoSavedFile("designLib", "testing1", "layout1");
```

Returns TRUE or FALSE

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

dbHasPanicFile

Boolean

```
dbHasPanicFile(  
    const char    *libName,  
    const char    *cellName,  
    const char    *viewName  
);
```

Description

Determines whether the specified cellview has a panic file.

Arguments

<i>libName</i>	Name of a library.
<i>cellName</i>	Name of a cellview.
<i>viewName</i>	Name of a view.

Return Value

<i>Boolean</i>	TRUE if the panic file is present in the specified cellview. FALSE if the panic file is not present in the specified cellview.

Example

```
Boolean result = dbHasPanicFile("designLib", "testing1", "layout1");
```

Returns TRUE or FALSE

dbRestoreAndOpenAutoSavedFile

```
dbCellViewId  
dbRestoreAndOpenAutoSavedFile(  
    const char    *libName,  
    const char    *cellName,  
    const char    *viewName  
);
```

Description

Restores and opens the auto-saved file for the specified cellview.

Arguments

<i>libName</i>	Name of a library.
<i>cellName</i>	Name of a cellview.
<i>viewName</i>	Name of a view.

Return Value

<i>dbCellViewID</i>	Database ID of the cellview that opens. Otherwise, returns NULL.

Example

```
dbCellViewId cvId = RestoreAndOpenAutoSavedFile ("designLib", "testing1",  
"layout1");
```

Returns the database ID of the cellview that opens, or returns NULL on error.

dbRestoreAndOpenPanicFile

```
dbCellViewId  
dbRestoreAndOpenPanicFile(  
    const char    *libName,  
    const char    *cellName,  
    const char    *viewName  
);
```

Description

Restores and opens the panic file for the specified cellview.

Arguments

<i>libName</i>	Name of a library.
<i>cellName</i>	Name of a cellview.
<i>viewName</i>	Name of a view.

Return Value

<i>dbCellViewID</i>	Database ID of the cellview that opens. Otherwise, returns NULL.

Example

```
dbCellViewId cvId = RestoreAndOpenPanicFile ("designLib", "testing1", "layout1");
```

Returns database ID of the cellview that opens or returns NULL on error.

Hierarchy Traversal Functions

The following functions traverse the hierarchy to retrieve information about it.

dbOpenHier

```
void
dbOpenHier(
    dbCellViewId cellViewId, unsigned int nLevels, FuncBooleanPtr msgConsumer
);
```

Opens `nLevels` of descendants of `cellViewId`, and brings them into core. If it cannot bring a descendant into core, it calls `msgConsumer` and checks the returned value. If the returned value from `msgConsumer` is **FALSE**, then `dbOpenHier` stops immediately. The `msgConsumer` function is defined as follows:

```
Boolean
msgConsumer(instHeader, errorMessage
    dbInstHeaderId    instHeader;
    String            errorMessage;
```

dbOpenAllMaster

```
Boolean
dbOpenAllMaster(
    dbCellViewId cellViewId
);
```

Opens all master cellviews referred to by this cellview and binds them to the corresponding instance headers. This is equivalent to

```
dbOpenHier(cellViewId, dbcMaxHier, NULL)
```

Operations on Cellviews

A cellview is a database object despite its special function as a container of all other objects. It has attributes, properties, and relationships. The `one-to-many` relationships associated with a cellview let you traverse the entire cellview and visit every object in the database. The relationships and the generators are documented separately in this manual.

Cellview Attributes Functions

The following functions return information on cellview attributes.

dbGetCellViewLibId

```
ddId  
dbGetCellViewLibId(  
    dbCellViewId cellViewId  
);
```

Returns the library containing the given cellview.

dbGetCellId

```
ddId  
dbGetCellId(  
    dbCellViewId cellViewId  
);
```

Returns the `ddId` of the cell for the specified cellview.

dbGetViewId

```
ddId  
dbGetViewId(  
    dbCellViewId cellViewId  
);
```

Returns the `ddId` of the view for the specified cellview.

dbGetCellViewView

```
ddId
```

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

```
dbGetCellViewView(  
    dbCellViewId cellViewId  
);
```

Returns the `ddId` of the view for the specified cellview.

dbGetCellViewCellName

```
String  
dbGetCellViewCellName(  
    dbCellViewId cellViewId  
);
```

Returns the cell name of the cellview.

dbGetCellViewViewName

```
String  
dbGetCellViewViewName(  
    dbCellViewId cellViewId  
);
```

Returns the view name of the cellview.

dbGetCellViewMode

```
String  
dbGetCellViewMode(  
    dbCellViewId cellViewId  
);
```

Returns the mode of the cellview as a string. Valid modes are `w` (write), `r` (read), and `a` (append).

dbGetCellViewFullDirPath

```
String  
dbGetCellViewFullDirPath(  
    dbCellViewId cellViewId  
);
```

Returns the full path name of the cellview as a string.

dbGetCellViewBBox

```
dbBBox *  
dbGetCellViewBBox(  
    dbCellViewId cellViewId  
);
```

Returns the bounding box for the cellview. This bounding box is derived by taking the union of the bounding boxes of all objects in the cellview (that is, a bounding box just big enough to contain the bounding boxes of all the objects).

Refer to [“Bounding Boxes”](#) on page 132 for an explanation of the bounding box concept.

dbGetCellViewCellType

Boolean

```
dbGetCellViewCellType(  
    dbCellViewId  cvId,  
    dbCellType*   cellType  
);
```

Description

Returns the cell type for the specified cellview.

Arguments

cvId ID of a cellview.

dbCellType Returned type of the cell.

Possible return values: dbcCellTypeNone,
dbcCellTypeBlock, dbcCellTypeBlockRing,
dbcCellTypeCover, dbcCellTypeCoverBump,
dbcCellTypePad, dbcCellTypePadSpacer,
dbcCellTypeCore, dbcCellTypeCoreSpacer,
dbcCellTypeCoreAntenna, dbcCellTypeCorner,
dbcCellTypeSoftMacro, dbcCellTypeVia,
dbcCellTypeBlockBlackBox,
dbcCellTypePadAreaIO, dbcCellTypeCoreWellTap

Return Value

Boolean TRUE, if the cell type for the specified cellview is returned.
FALSE, if the operation failed.

Example

```
// This program will print "The cellType is core"  
  
    dbCellViewId    cvId;  
    bool            result = true;  
  
    if (!(cvId = dbOpenCellViewByType(argv[1], argv[2], argv[3], "maskLayout",  
(char*)"w", (dbCellViewId)NULL))) {
```

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

```
        fprintf(stderr, "ERROR: Unable to open %s/%s/%s\n", argv[1], argv[2],
argv[3]);
        dbErrorFlushWarn();
        return false;
    }

    dbCellType cellType = dbcCellTypeUnknown;

    if (dbSetCellViewCellType(cvId, dbcCellTypeCore)) {
        if (dbGetCellViewCellType(cvId, &cellType)) {
            fprintf(stdout, "The cellType is %s\n", dbCellTypeToString(cellType));
        } else {
            fprintf(stderr, "ERROR: Unable to get the cellType of the cellView\n");
            result = false;
        }
    } else {
        fprintf(stderr, "ERROR: Failed to set the cellType of the cellView\n");
        result = false;
    }
}
```

Cellview Voltage Range Functions

dbGetCellViewNetVoltageRange

Boolean

```
dbGetCellViewNetVoltageRange(  
    dbCellViewId cvId,  
    float *minVoltage,  
    float *maxVoltage  
);
```

Description

Returns the default minimum and maximum net voltages from the specified cellview ID.

Arguments

<i>cvId</i>	The database cellview ID whose minimum and maximum default net voltages must be returned.
<i>minVoltage</i>	The minimum default net voltage.
<i>maxVoltage</i>	The maximum default net voltage.

Return Value

Boolean	Returns TRUE if the command is successful; otherwise returns FALSE.

dbIsCellViewNetVoltageRangeSet

Boolean

```
dbIsCellViewNetVoltageRangeSet(  
    dbCellViewId cellViewId  
);
```

Description

Ensures that the voltage range has been set on a cellview.

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

Arguments

cellViewId Database ID of a cellview.

Return Value

Boolean Returns TRUE if the voltage range has been set on the specified cellview; otherwise returns FALSE.

dbSetCellViewNetVoltageRange

Boolean

```
dbSetCellViewNetVoltageRange(  
    dbCellViewId cvId,  
    float minVoltage,  
    float maxVoltage  
);
```

Description

Sets the default minimum and maximum net voltages on a cellview ID.

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

Arguments

<i>cvId</i>	The database cellview ID whose minimum and maximum default net voltages must be returned.
<i>minVoltage</i>	The minimum default net voltage.
<i>minVoltage</i>	The maximum default net voltage.

Return Value

<i>Boolean</i>	Returns TRUE if the command is successful; otherwise returns FALSE.

dbUnsetCellViewNetVoltageRange

Boolean

```
dbUnsetCellViewNetVoltageRange(  
    dbCellViewId cellViewId  
);
```

Description

Resets the voltage range on a cellview to its default value.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

Boolean	Returns TRUE if the voltage range is reset; otherwise returns FALSE.

Predicate Function for Cellviews

The following function returns predicate (Boolean) information on cellviews.

dbIsCellViewModified

Boolean

```
dbIsCellViewModified(  
    dbCellViewId cellViewId  
);
```

Returns TRUE if the cellview was modified but not yet saved on disk. Otherwise, returns FALSE.

Cellview Attributes of Purpose, Type, and Status

Each cellview object has three other attributes associated with it: `cellViewType`, `cellViewStatus`, and `cellViewPurpose`. These attributes describe how CDBA uses the information in the cellview.

Although the entire spectrum of enumerated constant values for these attributes is presented in the tables that follow, only a few values are currently defined and supported in Cadence® application programs. These attributes are of primary interest to placement and routing applications and can be ignored by other application programs.



Caution

The names and meanings of these attributes might change in future versions of CDBA. For now, only the enumerated constants currently defined for cellview type, status, and purpose are documented.

Predefined Cellview Purposes

Name	Enumerated constants in type dbCellViewPurpose
Drawing	dbcDrawingCVP
Pin	dbcPinCVP
Net	dbcNetCVP
Boundary	dbcBoundaryCVP
Plane	dbcPlaneCVP
Stub	dbcStubCVP
Changed	dbcChangedCVP
Contact	dbcContactCVP
Active Area	dbcActiveAreaCVP
Multiple Pins	dbcMultiPinCVP
Cell	dbcCellCVP
All	dbcAllCVP

Predefined Cellview Types

Enumerated constants in type dbCellviewType	Explanation
dbcGraphic	Graphics and text of displayed data
dbcMaskLayout	Integrated circuit layout
dbcSchematic	Schematic representation of connectivity of a cell
dbcSymbolic	Symbolic layout—a shorthand layout description
dbcPCB	Printed circuit board layout

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

Predefined Cellview Types

Enumerated constants in type <code>dbCellviewType</code>	Explanation
<code>dbcSchematicSymbol</code>	Logic symbol
<code>dbcLogicModel</code>	Logic simulation data
<code>dbcBehavior</code>	Behavior of a cell
<code>dbcStranger</code>	An “as yet unknown” representation of a cell
<code>dbcNetList</code>	Netlist

Predefined Cellview Statuses

Enumerated constants in type `dbCellviewStatus`

`dbcFluid`
`dbcBounded`
`dbcInsidePlaced`
`dbcRouteReady`
`dbcWithFeedthroughs`
`dbcGlobalRouted`
`dbcRouted`
`dbcFinished`
`dbcStickDone`

When a cellview is created, all of its cellview attributes are initialized from the corresponding template cellview.

Cellview Type and Status Functions

The following functions retrieve and set information about the cellview type and status.

dbGetCellViewType

```
dbCellViewType  
dbGetCellViewType(  
    dbCellViewId cellViewId  
);
```

Returns the type of the cellview.

dbGetCellViewStatus

```
dbCellViewStatus  
dbGetCellViewStatus(  
    dbCellViewId cellViewId  
);
```

Returns the status of the cellview.

dbSetCellViewStatus

```
Boolean  
dbSetCellViewStatus(  
    dbCellViewId cellViewId, dbCellViewStatus status  
);
```

Sets the status of the cellview.

dbIsCellViewPhysicalOnly

Boolean

```
dbIsCellViewPhysicalOnly(  
    dbCellViewId cvId  
);
```

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>cvId</i>	Database ID of a cellview.

Return Value

<i>Boolean</i>	TRUE if the cellview is visible to block domain only. FALSE if the cellview is visible to both block and module domain.

dbUnsetCellViewPhysicalOnly

Boolean

```
dbUnsetCellViewPhysicalOnly(  
    dbCellViewId cvId  
);
```

Description

Unsets the physical-only attribute of the specified cellview, allowing connectivity to be visible in the OpenAccess module domain.

Arguments

<i>cvId</i>	Database ID of a cellview.

Return Value

<i>Boolean</i>	TRUE if the physical-only attribute of the specified cellview is unset. FALSE if the operation fails.

Counter Functions

In OpenAccess, the scheme to designate that the database has been edited is through updating integer counters (unsigned). Counters in are OpenAccess are termed time stamps, but their values are unsigned integers.

In addition to maintaining a counter for the cellview, OpenAccess maintains counters for many other objects in the database. Whenever an edit is made, any existing counter for the object plus the counter for the cellview are increased by 1.

dbIsConnCurrent

Boolean

```
dbIsConnCurrent (
    dbCellViewId    cellViewId
)
```

Description

Tests whether the value of the connectivity current property is equal to the cellview timestamp counter value.

Arguments

<code>cellViewId</code>	The database <code>dbCellViewId</code> for the cell view.

Return Value

Boolean	Returns TRUE if the value of the connectivity current property is equal to the cellview's timestamp counter value. Returns FALSE if 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.

dbDisableCellViewCounter

```
void  
dbDisableCellViewCounter(  
    dbCellViewId    cellViewId  
)
```

Description

Disables updating of the cellview timestamp counter.

Arguments

<i>cellViewId</i>	The database dbCellViewId for the cellview.

Return Value

Void.

dbGetCellViewCounter

Boolean

```
dbGetCellViewCounter(  
    dbCellViewId    cellViewId  
)
```

Description

Gets the last saved time for the cellview timestamp counter.

Arguments

<code>cellViewId</code>	The database <code>dbCellViewId</code> for the cellview.

Return Value

Boolean	Returns TRUE if the last saved time for the cellview timestamp counter was successfully retrieved. Returns FALSE otherwise.

dbEnableCellViewCounter

```
void  
dbEnableCellViewCounter(  
    dbCellViewId    cellViewId  
)
```

Description

Enables updating of the cellview timestamp counter.

Arguments

<code>cellViewId</code>	The database <code>dbCellViewId</code> for the cellview.

Return Value

`void`

dbSetConnCurrent

```
void  
dbSetConnCurrent(  
    dbCellViewId    cellViewId  
)
```

Description

Sets the value of the connectivity current property to the cellview timestamp counter value.

Arguments

<code>cellViewId</code>	The database <code>dbCellViewId</code> for the cellview.

Return Values

`void`

dbGetFileTimeStamp

```
time
dbGetFileTimeStamp (
    dbCellViewId    cellViewId
)
```

Description

Gets the last saved time for the cellview file.

Arguments

<code>cellViewId</code>	The database <code>dbCellViewId</code> for the cellview.

Return Values

<code>time</code>	The last saved time cellview file.

dbSetInstHeadersCounter

Boolean

```
dbSetInstHeadersCounter(  
    dbCellViewId    cellViewId  
)
```

Description

Sets the value of the cellview instHeaders counter.

Arguments

cellViewId	The database dbCellViewId for the cellview.

Return Values

Boolean	Returns TRUE if the cellview counter value was successfully set in the instHeaders counter. Returns FALSE otherwise.

Database I/O Functions

The following functions retrieve cellview information from the database and modify cellview information in the database.

dbRefreshCellView

Boolean

```
dbRefreshCellView(  
    dbCellViewID cellViewId  
);
```

Refreshes the cellview in virtual memory with the copy from disk. Any unsaved changes to the cellview are lost. The function purges the cellview from virtual memory and reopens the cellview append mode. If any errors occur, the function returns FALSE.

dbWriteCellView

Boolean

```
dbWriteCellView(  
    dbCellViewId cellViewId, String libName, String cellName, String viewName,  
    String version, dbCellViewId contextCellViewId  
);
```

Writes cellview `cellViewId` to disk under the specified cell and view names. The version requirement is for backward compatibility and must always be specified as NULL.

dbCopyCellView

dbCellViewId

```
dbCopyCellView(  
    dbCellViewId cellViewId, String libName, String cellName, String viewName,  
    String version, dbCellViewId contextCellViewId, Boolean overwrite  
);
```

Copies cellview `cellViewId` to a destination cellview of the specified cell, view, and version names. The function modifies both the disk version and the virtual memory version of the destination cellview. If the copy succeeds, the function returns TRUE. The version requirement is for backward compatibility and must always be specified as NULL.

`dbCopyCellView` and `dbWriteCellView` will copy all the files defined in the masterSet part in the data registry file. For example, if the file `virtuoso.reg` for `maskLayout` is like:

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

```
DataFormat MaskLayout {  
    Pattern=layout.oa;  
    Default_Editor=Virtuoso;  
    dfII_ViewType=maskLayout;  
    Co_Managed= layout.oa master.tag data.dm pc.db *.att verilog.v verilog.vams  
                ams_direct.dat amsAPT.apl es.db esb.db *.conf *.ctstch *.mode  
                enc.pref.tcl SiFix.option *.fp *.fp.pg *.rcdb.gz *.ymp;  
    Master_Set= layout.oa master.tag pc.db;  
    updaterProg=ccpDFIUPdtr;  
    checkerProg=ccpDFIIChecker  
}
```

Then dbCopyCellView and dbWriteCellView will copy "layout.oa master.tag pc.db" all together.



Both dbCopyCellView and dbWriteCellView force the destination cellview to be purged from virtual memory. Therefore, if the destination cellview has been opened before either dbCopyCellView or dbWriteCellView is called, the ID of the destination cellview will become invalid after the copy or write operation.

Cadence Integrators Toolkit Database Reference

CDBA Input and Output Functions

CDBA Figure Functions

A figure is a class of database objects that includes instances, geometric shapes, and mosaics. Figures are accessible from nets or pins and are the geometries or instances that implement a net or pin. A net without figures is invisible when viewed with the graphics editor. A pin cannot exist without a figure.

This chapter discusses the following:

- [Figure Class Hierarchy](#) on page 209
- [Blockages](#) on page 215
- [Area Halos](#) on page 246
- [Layer Halos](#) on page 263
- [Boundaries](#) on page 272
- [Clusters](#) on page 313
- [LayerHeaders](#) on page 316
- [Markers](#) on page 335
- [RowHeaders](#) on page 369
- [EEQ Masters](#) on page 381
- [EEQ Masters](#) on page 381
- [Track Patterns](#) on page 397
- [Shape Figure Functions](#) on page 412
- [Shape Route Status](#) on page 450
- [Instance and Hierarchy Functions](#) on page 463
- [FigGroup Functions](#) on page 513
- [Region Queries](#) on page 520
- [Parameterized Cells](#) on page 563

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

- [Abutment Attributes](#) on page 571
- [Mosaic Functions](#) on page 579
- [Guides](#) on page 594
- [pathSegs](#) on page 605
- [Vias](#) on page 623
- [Routes](#) on page 647
- [CDBA Connectivity Functions](#) on page 679
- [Nets](#) on page 684
- [Signals](#) on page 705
- [Terminals](#) on page 711
- [Instance Terminals](#) on page 740
- [Pins](#) on page 748
- [Access Directions of a Pin](#) on page 764
- [BusDef and BitOrder](#) on page 766
- [Cellview Information](#) on page 798
- [Equivalent Connectivity Time Functions](#) on page 803
- [Net and Terminal Name Syntax](#) on page 808
- [Name and Vector Expression Expansion](#) on page 810
- [Automatic Database Deletions](#) on page 811

Figure Class Hierarchy

The class hierarchy for figures is shown in the following table.

Class Hierarchy for Figure

Name	Enumerated constant	ID type for type dbType
Instance	dbcInstType	dbInstId
Shape	-	dbShapeId
Rectangle	dbcRectType	dbRectId
Polygon	dbcPolygonType	dbPolygonId
Line	dbcLineType	dbLineId
Path	dbcPathType	dbPathId
Arc	dbcArcType	dbArcId
Ellipse	dbcEllipseType	dbEllipseId
Label	dbcLabelType	dbLabelId
Donut	dbcDonutType	dbDonutId
Dot	dbcDotType	dbDotId
TextDisplay	dbcTextDisplayType	dbTextDisplayId
Mosaic	dbcMosaicType	dbMosaicId

Operations on Figures

The functions described in this chapter perform the following operations on figures:

- Retrieval and traversal operations
- Creation and deletion operations
- Move and copy operations

Retrieval and Traversal Operations

The following functions traverse the hierarchy to retrieve or change information about a figure.

dbIsFigId

Boolean

```
dbIsFigId(  
    dbFigId figId  
);
```

Returns TRUE if the ID passed in as `figId` is a figure. Otherwise, it returns FALSE.

dbGetFigPurpose

dbPurpose

```
dbGetFigPurpose(  
    dbFigId figId  
);
```

Retrieves the purpose of the figure specified by `figId`. The purpose can be either a reserved purpose (for example, `dbcCellPurpose`, `dbcNetPurpose`) or a user-defined purpose.

dbSetFigPurpose

Boolean

```
dbSetFigPurpose(  
    dbFigId figId, dbPurpose purpose  
);
```

Sets the purpose of the figure specified by `figId`. The purpose can be either a reserved purpose (for example, `dbcCellPurpose`, `dbcNetPurpose`) or a user-defined purpose.

dbAddFigToNet

Boolean

```
dbAddFigToNet(  
    dbFigId figId, dbNetId netId  
);
```

Adds (or attaches) a figure to a net.

dbSubFigFromNet

Boolean

```
dbSubFigFromNet(dbFigId figId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Subtracts (or detaches) a figure from a net.

Because pins must have only one figure attached to them, use `dbCreatePin` and `dbDeletePin` to add the figure corresponding to a net or to subtract the figure. A figure already attached to another net or pin must be detached before being added to this net. A figure's purpose remains the same after it is added or subtracted from a net.

dbSubFigFromPin

Boolean

```
dbSubFigFromPin(  
    dbFigId figId  
);
```

Subtracts (or detaches) a figure from a pin. A figure already attached to another net or pin must be detached before being added to this pin.

The pin that the figure implements is automatically deleted. The terminal that contains the pin is not deleted. A figure's purpose remains the same after it is subtracted from a pin.

dbGetFigNet

dbNetId

```
dbGetFigNet(  
    dbFigId figId  
);
```

Returns the net that contains the figure specified by `figId`. If the figure doesn't belong to a net, it returns `dbcNullId`.

dbGetFigPin

dbPinId

```
dbGetFigPin(  
    dbFigId figId  
);
```

Returns the pin that contains the figure specified by `figId`. If the figure doesn't belong to a pin, it returns `dbcNullId`.

dbGetFigBBoxf

dbBBox *

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetFigBBox(  
    dbFigId figId  
);
```

Returns the bounding box of the figure specified by `figId`.

dbStartGenFig and dbGenFig

```
dbGenStateId  
  
dbStartGenFig(  
    dbCellViewId cellViewId  
);  
  
Boolean  
  
dbGenFig(  
    dbGenStateId stateId, dbFigId pFigId) /* modifies FigId */
```

These functions generate all figures (that is, instances, mosaics, and shapes) contained in the cellview specified by `cellViewId`. These functions generate parent as well as child figures.

dbStartGenNonChildFig and dbGenNonChildFig

```
dbGenStateId  
  
dbStartGenNonChildFig(  
    dbCellViewId cellViewId  
);  
  
Boolean  
  
dbGenNonChildFig(  
    dbGenStateId stateId, dbFigId pFigId  
);
```

These functions generate all figures contained in the cellview specified by `cellViewId` except child figures.

The following retrieval and traversal functions for figures are documented in [CDBA Connectivity Functions](#) on page 679:

- [dbGetPinFig](#) on page 752
- [dbNetHasFig](#) on page 696
- [dbStartGenNetToFig](#) on page 700

Creation and Deletion Operations

There is currently only a function to delete a figure:

dbDeleteFig

Boolean

```
dbDeleteFig(  
    dbFigId figId  
);
```

Deletes the figure specified by `figId`. If the figure is a parent, the descendants are deleted as well. (See [Chapter 9, “CDBA Parent-Child Relationship Functions.”](#) for more information on deleting a figure.) After `dbDeleteFig` is called, `figId` is invalid and must not be used. Any identifiers to child figures are likewise invalid.

Move and Copy Operations

dbMoveFig

dbFigId

```
dbMoveFig(  
    dbFigId figId, dbCellViewId destCellViewId, dbTransform transform  
);
```

Moves the figure specified by `figId`. The figure is magnified, rotated, and reflected about the origin as specified by the transformation's orientation. It is then translated by the transformation's offset. If the figure, cellview, or transformation is invalid, it gives a fatal error message and returns `dbcNullFigId`. Otherwise, it returns the ID of the transformed figure.

If `destCellViewId` is `dbcNullCellViewId` or if `destCellViewId` is the cellview that contains the figure, the figure moves within the same cellview and the figure's ID remains the same. The figure's purpose and properties remain the same.

If the `destCellViewId` is not the original figure's cellview, the figure is deleted, and a new figure is created in the cellview specified by `destCellViewId`. The newly created figure has a different ID and retains the purpose and properties of the original figure. The new figure is not associated with any net, pin, or terminal of the original figure.

If `figId` refers to a parent, then all of the figure's descendants are moved by the same transformation. This transformation also applies when moving between cellviews.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

See [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#) for more information on moving a figure.

dbCopyFig

dbFigId

```
dbCopyFig(  
    dbFigId figId, dbCellViewId destCellViewId, dbTransform transform  
);
```

Copies the figure specified by `figId`. The copied figure is rotated and reflected about the origin as specified by the transformation's orientation. Then it is translated by the transformation's offset. If the figure, cellview, or transformation is invalid, it gives a fatal error message and returns `dbcNullFigId`. Otherwise, it returns the ID of the copied figure.

If `destCellViewId` is `dbcNullCellViewId` or if `destCellViewId` is the cellview that contains the figure, the figure is copied in the same cellview. If `destCellViewId` is not `dbcNullCellViewId` or the original figure's cellview, the figure is copied into the cellview specified by `destCellViewId`.

The new figure retains the purpose and properties of the original figure. The figure is not associated with any net, pin, or terminal of the original figure.

If `figId` refers to a parent, then all of the figure's descendants are copied by the same transformation and placed into the appropriate cellview. Copied children retain their purpose and properties. See [CDBA Parent-Child Relationship Functions](#) for more information on copying a figure.

Blockages

This section describes functions that affect blockage objects.

About Blockages

A blockage is a figure object that indicates an area on which other objects may not be placed. The blockage area is specified by its point array.

Each blockage has a type that indicates the type of object the blockage blocks.

```
typedef enum dbBlockageType {
    dbcUnknownBlockageType = 0,
    dbcRoutingBlockage      = 1,
    dbcViaRoutingBlockage   = 2,
    dbcPlacementBlockage    = 3,
    dbcWiringBlockage       = 4,
    dbcFillBlockage         = 5,
    dbcSlotBlockage         = 6,
    dbcPinBlockage          = 7,
    dbcFeedthruBlockage     = 8,
    dbcScreenBlockage       = 9
} dbBlockageType;

const String dbvBlockageTypeString [] = {
    "unknown",
    "routing",
    "viaRouting",
    "placement",
    "wiring",
    "fill",
    "slot",
    "pin",
    "feedthru",
    "screen"
};
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsBlockageId

Boolean

```
dbIsBlockageId(  
    dbBlockageId    blockageId  
);
```

Description

Checks that the *blockageId* refers to a blockage object.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>Boolean</i>	Returns TRUE if the <i>blockageId</i> refers to a valid blockage object. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteBlockage

Boolean

```
dbDeleteBlockage(  
    dbBlockageId    blockageId  
);
```

Description

Deletes the blockage object specified by `blockageId`.

Arguments

<code>blockageId</code>	The database identifier of the blockage to be deleted.

Return Values

Boolean	Returns TRUE if the blockage object was successfully deleted. Returns FALSE otherwise.

dbIsBlockageOrthogonal

Boolean

```
dbIsBlockageOrthogonal(  
    dbBlockageId    blockageId  
);
```

Description

Checks that the blockage specified by the database `blockageId` is orthogonal.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>Boolean</i>	Returns TRUE if the blockage specified by <code>blockageId</code> is orthogonal. Returns FALSE otherwise.

dbIsBlockageRectangle

Boolean

```
dbIsBlockageRectangle(  
    dbBlockageId    blockageId  
);
```

Description

Checks that the blockage specified by the database `blockageId` is a rectangle.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>Boolean</i>	Returns TRUE if the blockage specified by <code>blockageId</code> is a rectangle. Returns FALSE otherwise.

dbIsBlockagePushedDown

Boolean

```
dbIsBlockagePushedDown(  
    dbBlockageId      blockageId  
);
```

Description

Checks that the blockage specified by the database `blockageId` is pushed down from a higher cellview.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>Boolean</i>	Returns TRUE if the blockage specified by <code>blockageId</code> was pushed down from a higher cellview. Returns FALSE otherwise.

dbSetBlockagePushedDown

Boolean

```
dbSetBlockagePushedDown(  
    dbBlockageId      blockageId,  
    Boolean            pushedDown  
);
```

Description

Sets the pushed down attribute for the blockage specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.
<i>pushedDown</i>	Sets the <code>pushedDown</code> field for the specified blockage.

Return Values

Boolean	Returns TRUE if the <code>pushedDown</code> field for the specified blockage was successfully set. Returns FALSE otherwise.

dbGetBlockageType

```
dbBlockageType  
dbGetBlockageType (  
    dbBlockageId      blockageId  
);
```

Description

Gets the blockage type specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>dbBlockageType</i>	Gets the type field for the specified blockage. The default is unknown. The valid enum choices are: dbcRoutingBlockage dbcViaRoutingBlockage dbcPlacementBlockage dbcWiringBlockage dbcFillBlockage dbcSlotBlockage

dbLayerBlockageHasEffectiveWidth

Boolean

```
dbLayerBlockageHasEffectiveWidth(  
    dbLayerBlockageId    layerBlockageId  
);
```

Description

Checks that the effective width is set for the layer blockage.

Arguments

<i>layerBlockageId</i>	The database identifier of the layer blockage object.

Return Values

Boolean	Returns TRUE if the effective width is set for the blockage specified by <i>layerBlockageId</i> . Returns FALSE otherwise.

dbGetLayerBlockageEffectiveWidth

```
dbDistance  
dbGetLayerBlockageEffectiveWidth(  
    dbBlockageId    blockageId  
);
```

Description

Gets the effective width in database units (DBU) of the layer blockage specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the layer blockage object.

Return Values

<code>dbDistance</code>	Returns the effective width in DBU of the layer blockage.

dbGetBlockageDensity

```
float  
dbGetBlockageDensity(  
    dbBlockageId    blockageId  
);
```

Description

Gets the density of the blockage specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<code>float</code>	Returns the density of the blockage. If blockage density is not specified, returns 100.0.

dbSetBlockageDensity

Boolean

```
dbSetBlockageDensity(  
    dbBlockageId    blockageId,  
    float            density  
);
```

Description

Sets the density of the blockage specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.
<i>density</i>	The density of the blockage. The density value must be in [0, 100.0].

Return Values

Boolean	Returns TRUE if the blockage density is set successfully. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetBlockageOwner

```
dbId  
dbGetBlockageOwner(  
    dbBlockageId    blockageId  
);
```

Description

Gets the owner for the blockage specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<code>dbId</code>	Returns the owner of the blockage specified by <code>blockageId</code> .

dbSetBlockageOwner

Boolean

```
dbSetBlockageOwner(  
    dbBlockageId    blockageId,  
    dbId            ownerId  
);
```

Description

Sets the owner for the blockage specified by the database `blockageId`.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.
<i>ownerId</i>	The owner to be set for the blockage specified by <code>blockageId</code> .

Return Values

Boolean	Returns TRUE if the owner is successfully set for the blockage specified by <code>blockageId</code> . Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetBlockage

Boolean

```
dbGetBlockage (
    dbBlockageId      blockageId,
    unsigned int       *nPoints,
    dbPoint            **pointArray
);
```

Description

Gets the points for the blockage specified by the database *blockageId*.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.
<i>*nPoints</i>	The pointer to integer to be filled with the number of points for the blockage.
<i>**pointArray</i>	The pointer to <i>dbPoint</i> array to be filled with the points for the blockage.

Return Values

Boolean	Returns TRUE if the points are successfully retrieved for the blockage specified by <i>blockageId</i> . Returns FALSE otherwise.

dbGetBlockageArea

```
double  
dbGetBlockageArea(  
    dbBlockageId    blockageId  
);
```

Description

Gets the area in database units (DBU) squared for the blockage specified by the database *blockageId*.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>double</i>	Returns the area in DBU squared of the blockage specified by <i>blockageId</i> .

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsAreaBlockageId

Boolean

```
dbIsAreaBlockageId(  
    dbId      objId  
);
```

Description

Verifies that the *objId* refers to a valid area blockage object.

Arguments

<i>objId</i>	The database identifier of the blockage object.

Return Values

Boolean	Returns TRUE if the <i>objId</i> is a valid area blockage object. Returns FALSE otherwise.

dbCreateAreaBlockage

```
dbAreaBlockageId  
dbCreateAreaBlockage(  
    dbCellviewId    cvId,  
    dbId            ownerId,  
    unsigned int     nPoints,  
    const dbPoint    *pointArray  
);
```

Description

Creates an area blockage object in the cellview. The area blockage created is of type placement.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>ownerId</i>	The owner of the area blockage. The owner can be NULL, or it can be an instance or a cluster. Any other object type as owner causes area blockage creation to fail with a warning message.
<i>nPoints</i>	The number of points in the area blockage.
<i>*pointArray</i>	The points that define the area blockage.

Return Values

<i>dbAreaBlockageId</i>	Returns the <i>dbAreaBlockageId</i> of the created area blockage object.

dbSetAreaBlockagePoint

Boolean

```
dbSetAreaBlockagePoint(  
    dbAreaBlockageId    areaBlockageId,  
    unsigned int         npoints,  
    const dbPoint        *pointArray  
);
```

Description

Sets the points of the area blockage.

Arguments

<i>areaBlockageId</i>	The database identifier of the area blockage.
<i>npoints</i>	The number of points in the area blockage.
<i>*pointArray</i>	The points that define the area blockage.

Return Values

Boolean	Returns TRUE if the points are set successfully. Returns FALSE otherwise.

dbTransformAreaBlockage

Boolean

```
dbTransformAreaBlockage (
    dbAreaBlockageId    areaBlockageId,
    double               scale,
    double               angle
);
```

Description

Transforms the area blockage by the scale and angle.

Arguments

<i>areaBlockageId</i>	The database identifier of the area blockage.
<i>scale</i>	The scale to which to magnify the area blockage.
<i>angle</i>	The angle at which to rotate the area blockage.

Return Values

Boolean	Returns TRUE if the area blockage is transformed successfully. Returns FALSE otherwise.

dbIsLayerBlockageId

Boolean

```
dbIsLayerBlockageId(  
    dbId      objId  
);
```

Description

Verifies that the *objId* refers to a valid layer blockage object.

Arguments

<i>objId</i>	The database identifier of the layer blockage.

Return Values

Boolean	Returns TRUE if the <i>objId</i> refers to a valid layer blockage object. Returns FALSE otherwise.

dbCreateLayerBlockage

```
dbLayerBlockageId  
dbCreateLayerBlockage(  
    dbCellViewId    cvId,  
    dbLayer          layerNum,  
    dbBlockageType   type,  
    dbId             ownerId,  
    unsigned int     nPoints,  
    const dbPoint    *pointArray  
);
```

Description

Creates a layer blockage object in the cellview.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>layerNum</i>	The layer number of the blockage. If the layer number is unknown or <code>dbcNullLayer</code> , an error results.
<i>type</i>	The type of the layer blockage. If the blockage type is placement, the function creates the layer blockage, but issues a warning.
<i>ownerId</i>	The owner of the layer blockage. The owner can be NULL, or it can be an instance or a cluster. Any other object type as owner causes area blockage creation to fail with a warning message.
<i>nPoints</i>	The number of points in the area blockage.
<i>*pointArray</i>	The points that define the area blockage.

Return Values

<i>dbLayerBlockageId</i>	Returns the <i>dbLayerBlockageId</i> of the created layer blockage object.

dbGetLayerBlockageLayerNum

```
dbLayer  
dbGetLayerBlockageLayerNum(  
    dbLayerBlockageId    layerBlockageId  
);
```

Description

Gets the layer number of the layer blockage.

Arguments

<i>layerBlockageId</i>	The database identifier of the layer blockage object.

Return Values

<i>dbLayer</i>	Returns the layer number of the layer blockage.

dbSetLayerBlockageLayerNum

Boolean

```
dbSetLayerBlockageLayerNum(  
    dbLayerBlockageId    layerBlockageId,  
    dbLayer              layerNum  
);
```

Description

Sets the layer number of the layer blockage.

Arguments

<i>layerBlockageId</i>	The database identifier of the layer blockage object.
<i>dbLayer</i>	The layer number of the layer blockage.

Return Values

Boolean	Returns TRUE if the layer number is set successfully. Returns FALSE otherwise.

dbGetLayerBlockageLayerHeader

```
dbLayerHeaderId  
dbGetLayerBlockageLayerHeader(  
    dbLayerBlockageId    layerBlockageId  
);
```

Description

Gets the layerHeader for the layer of the layerBlockageId.

Arguments

<i>layerBlockageId</i>	The database identifier of the layer blockage object.

Return Values

<i>dbLayerHeaderId</i>	Returns the layerHeader for the layer of the layer blockage.

dbSetLayerBlockagePoint

Boolean

```
dbSetLayerBlockagePoint(  
    dbLayerBlockageId    layerBlockageId,  
    unsigned int          nPoints,  
    const dbPoint         *pointArray  
);
```

Description

Sets the points of the `layerBlockageId`.

Arguments

<i>layerBlockageId</i>	The database identifier of the layer blockage object.
<i>nPoints</i>	Number of points in the layer blockage.
<i>*pointArray</i>	Points that define the layer blockage.

Return Values

Boolean	Returns TRUE if the points are set successfully. Returns FALSE otherwise.

dbTransformLayerBlockage

Boolean

```
dbTransformLayerBlockage (
    dbLayerBlockageId    layerBlockageId,
    double                scale,
    double                angle
);
```

Description

Transforms the layer blockage by the scale and angle.

Arguments

<i>layerBlockageId</i>	The database identifier of the layer blockage.
<i>scale</i>	Scale to which to magnify the layer blockage.
<i>angle</i>	Angle at which to rotate the layer blockage.

Return Values

Boolean	Returns TRUE if the layer blockage is transformed successfully. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenBlockage

```
dbGenStateId  
dbStartGenBlockage(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the blockages for a cellview.

Arguments

<i>cvId</i>	The database identifier of the cellview.

Return Values

<i>dbGenStateId</i>	Returns the generator <i>dbGenStateId</i> which keeps track of the state of blockage generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenBlockage

Boolean

```
dbGenBlockage (  
    dbGenStateId    pState,  
    dbBlockageId    *blockageId  
);
```

Description

Generates the next blockage in the cellview.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of blockage generation.
<i>*blockageId</i>	A pointer to the blockage ID. Returned if a blockage is generated.

Return Values

Boolean	Returns TRUE if a blockage is generated. Returns FALSE otherwise.

dbStartGenBlockageToPoint

```
dbGenStateId  
dbStartGenBlockageToPoint(  
    dbBlockageId    blockageId  
);
```

Description

Starts generating all the points for the given blockage.

Arguments

<i>blockageId</i>	The database identifier of the blockage object.

Return Values

<i>dbGenStateId</i>	Returns the generator <code>dbGenStateId</code> which keeps track of the state of point generation for the blockage.

dbGenBlockageToPoint

Boolean

```
dbGenBlockageToPoint(  
    dbGenStateId    pState,  
    dbPoint         *point  
);
```

Description

Generates the next point in the blockage.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of point generation.
<i>*point</i>	A pointer to the point. Returned if a point is generated.

Return Values

Boolean	Returns TRUE if a point is generated. Returns FALSE otherwise.

Area Halos

An area halo is a blockage. It is an extended area around an instance or a PR boundary in 4 orthogonal directions. The most common use is placement. The area halo keeps other instances from being placed within the halo.

The area halo has all the attributes of a blockage. In addition, an area halo has 4 offsets, one for each orthogonal direction.

A PR boundary halo is also known as a master halo. You can define an instance halo that is independent of the master halo. When both a PR boundary halo and an instance halo exist, use the instance halo. To determine whether a PR boundary has a master halo, query the PR boundary. An instance with no halo returns a NULL when queried even if a master halo exists.

Because an area halo is a blockage, any function that takes a `dbBlockageId` also applies to a `dbAreaHaloId`. The below blockage functions may apply when working with area halos.

- [dbGetBlockage](#) on page 229
- [dbGetBlockageType](#) on page 222
- [dbGetBlockageOwner](#) on page 227

The following generators generate area halos along with other types of blockages.

- [dbStartGenBlockage](#) on page 242
- [dbStartGenBlockageToPoint](#) on page 244
- [dbStartGenClusterToBlockage](#) on page 253
- [dbGenClusterToBlockage](#) on page 254
- [dbStartGenInstToBlockage](#) on page 255
- [dbGenInstToBlockage](#) on page 256
- [dbStartGenPRBoundaryToBlockage](#) on page 257
- [dbGenPRBoundaryToBlockage](#) on page 258

The `AreaHalo Id` is the following.

```
typedef dbId dbAreaHaloId;
```

The validation function verifies that the `dbAreaHaloId` is a valid `oaAreaHalo` object.

```
inline oaAreaHalo *  
dbiValidateAreaHalo (dbAreaHaloId areaHaloId);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsAreaHaloId

Boolean

```
dbIsAreaHaloId(  
    dbId      id  
);
```

Description

Verifies that the *id* refers to a valid area halo object.

Arguments

<i>id</i>	The database identifier of the area halo object to verify.

Return Values

Boolean	Returns TRUE if the <i>id</i> refers to an area halo object. Returns FALSE otherwise.

dbCreateAreaHalo

dbAreaHaloId

```
dbCreateAreaHalo(  
    dbId                ownerId,  
    dbDistance          topOffset,  
    dbDistance          bottomOffset,  
    dbDistance          leftOffset,  
    dbDistance          rightOffset  
);
```

Description

Creates an area halo with the specified attributes. The function validates the owner and checks the arguments.

The function first discovers if an area halo already exists on the owner. If a halo exists, the function issues a warning and returns a NULL ID. If no halo exists, the function creates an oaAreaHalo object on the owner and returns its database identifier.

Area halo is placement type blockage.

Arguments

<i>ownerId</i>	The owner of the area halo. Value must be either instance or PR boundary.
<i>topOffset</i>	Top offset of the area halo.
<i>bottomOffset</i>	Bottom offset of the area halo.
<i>leftOffset</i>	Left offset of the area halo.
<i>rightOffset</i>	Right offset of the area halo.

Return Values

dbAreaHaloId	Returns the area halo database identifier if no halo already exists on the owner. Returns a NULL ID and issues a warning if a halo does already exist on the owner.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteAreaHalo

Boolean

```
dbDeleteAreaHalo(  
    dbAreaHaloId    areaHaloId  
);
```

Description

Deletes the area halo.

Arguments

<i>areaHaloId</i>	The database identifier of the area halo object to delete.

Return Values

Boolean	Returns TRUE if the area halo is deleted. Returns FALSE if the <i>areaHaloId</i> is invalid.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindAreaHalo

```
dbAreaHaloId  
dbFindAreaHalo(  
    dbId          ownerId  
);
```

Description

Finds the placement area halo on the owner.

Arguments

<i>ownerId</i>	The database identifier of the area halo owner.

Return Values

Boolean	Returns the database identifier of the area halo. Returns a NULL database identifier if an area halo does not exist on the owner.

dbGetAreaHaloOffsets

Boolean

```
dbGetAreaHaloOffsets (
    dbAreaHaloId      areaHaloId,
    dbDistance         *topOffset,
    dbDistance         *bottomOffset,
    dbDistance         *leftOffset,
    dbDistance         *rightOffset
);
```

Description

Gets the offsets of the area halo. The function validates the area halo and fills the 4 offset pointers.

Arguments

<i>areaHaloId</i>	The database identifier of the area halo.
<i>*topOffset</i>	Top offset of the area halo.
<i>*bottomOffset</i>	Bottom offset of the area halo.
<i>*leftOffset</i>	Left offset of the area halo.
<i>*rightOffset</i>	Right offset of the area halo.

Return Values

Boolean	Returns TRUE if the area halo offset pointers are filled successfully. Returns FALSE otherwise.

dbSetAreaHaloOffsets

Boolean

```
dbSetAreaHaloOffsets (
    dbAreaHaloId      areaHaloId,
    dbDistance         topOffset,
    dbDistance         bottomOffset,
    dbDistance         leftOffset,
    dbDistance         rightOffset
);
```

Description

Sets the offsets of the area halo. The function validates the area halo and sets the 4 offsets.

Arguments

<i>areaHaloId</i>	The database identifier of the area halo.
<i>topOffset</i>	Top offset of the area halo.
<i>bottomOffset</i>	Bottom offset of the area halo.
<i>leftOffset</i>	Left offset of the area halo.
<i>rightOffset</i>	Right offset of the area halo.

Return Values

Boolean	Returns TRUE if the area halo offsets are set successfully. Returns FALSE otherwise.

dbStartGenClusterToBlockage

```
dbGenStateId  
dbStartGenClusterToBlockage (  
    dbClusterId      clusterId  
);
```

Description

Starts generating all the blockage objects owned by the cluster.

Arguments

<i>clusterId</i>	The database identifier of the cluster object.

Return Values

dbGenStateId	Returns the generator <code>dbGenStateId</code> which keeps track of the state of the blockage generation.

dbGenClusterToBlockage

Boolean

```
dbGenClusterToBlockage(  
    dbGenStateId    pState,  
    dbBlockageId    *blockageId  
);
```

Description

Generates the next blockage from the cluster.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of the blockage generation.
<i>*blockageId</i>	A pointer to the blockage ID. Returned if a blockage is generated.

Return Values

Boolean	Returns TRUE if a blockage is generated. Returns FALSE otherwise.

dbStartGenInstToBlockage

```
dbGenStateId  
dbStartGenInstToBlockage(  
    dbClusterId    clusterId  
);
```

Description

Starts generating all the blockage objects owned by the instance.

Arguments

<i>clusterId</i>	The database identifier of the cluster object.

Return Values

<i>dbGenStateId</i>	Returns the generator <i>dbGenStateId</i> which keeps track of the state of the blockage generation.

dbGenInstToBlockage

Boolean

```
dbGenInstToBlockage (
    dbGenStateId    pState,
    dbBlockageId    *blockageId
);
```

Description

Generates the next blockage from the instance.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of the blockage generation.
<i>*blockageId</i>	A pointer to the <code>dbBlockageID</code> . Returned if a blockage is generated.

Return Values

Boolean	Returns TRUE if a blockage is generated. Returns FALSE otherwise.

dbStartGenPRBoundaryToBlockage

```
dbGenStateId  
dbStartGenPRBoundaryToBlockage (  
    dbClusterId      clusterId  
);
```

Description

Starts generating all the blockage objects owned by the PR boundary.

Arguments

<i>clusterId</i>	The database identifier of the cluster object.

Return Values

dbGenStateId	Returns the generator <code>dbGenStateId</code> which keeps track of the state of the blockage generation.

dbGenPRBoundaryToBlockage

Boolean

```
dbGenPRBoundaryToBlockage (  
    dbGenStateId    pState,  
    dbBlockageId    *blockageId  
);
```

Description

Generates the next blockage from the PR boundary.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of the blockage generation.
<i>*blockageId</i>	A pointer to the <code>dbBlockageId</code> . Returned if a blockage is generated.

Return Values

Boolean	Returns TRUE if a blockage is generated. Returns FALSE otherwise.

dbCoverObstructionId

```
dbCreateCoverObstruction(dbPRBoundaryId boundaryId,  
                        dbIntRange      *maskRange,  
                        Boolean          allowPGNet);
```

Description

Creates a new cover obstruction object with the given `maskRange` and `allowPGNet` attributes. The cover obstruction is associated with the specified PR boundary object in a dependent way; this means if the PR boundary is deleted, its associated cover obstruction is also deleted.

dbDeleteCoverObstruction

```
Boolean  
    dbDeleteCoverObstruction(dbCoverObstructionIdcoverId);
```

Description

Deletes the cover obstruction object specified by `coverId`.

dbPRBoundaryId

```
dbGetCoverObstructionPRBoundary(dbCoverObstructionIdcoverId);
```

Description

Gets the PR boundary object associated with this cover obstruction object.

dbGetCoverObstructionPoints

```
Boolean  
dbGetCoverObstructionPoints(dbCoverObstructionIdcoverId,  
                            unsigned int *nPoints,  
                            dbPoint      **pointArray);
```

Description

Gets the points in the point array that defines this cover obstruction object.

dbGetCoverObstructionMaskRange

Boolean

```
dbGetCoverObstructionMaskRange (dbCoverObstructionIdcoverId,  
                                dbIntRange  
                                maskRange) ;
```

Description

Gets the mask range of cover obstruction object.

dbStartGenCoverObstructionToLayerHeader

dbGenStateId

```
dbStartGenCoverObstructionToLayerHeader (dbCoverObstructionIdcoverId) ;
```

Description

Starts the generator for all layer header objects whose corresponding mask number intersect the mask range of this cover obstruction object.

dbGenCoverObstructionToLayerHeader

Boolean

```
dbGenCoverObstructionToLayerHeader (dbGenStateId pState,  
                                     dbLayerHeaderId *layerHeaderId) ;
```

Description

Generates all layer header objects whose corresponding mask number intersect the mask range of this cover obstruction object.

dbGetCoverObstructionLayers

Boolean

```
dbGetCoverObstructionLayers (dbCoverObstructionId coverId,  
                             int          *nLayers,  
                             String       **layerNames);
```

Description

Gets all layer objects whose corresponding mask number intersect the mask range of this cover obstruction object.

dbGetCoverObstructionAllowPGNet

Boolean

```
dbGetCoverObstructionAllowPGNet (dbCoverObstructionId coverId);
```

Description

Returns a boolean indicating whether power and ground nets can be routed over the areas defined by this cover obstruction object.

dbSetCoverObstructionAllowPGNet

Boolean

```
dbSetCoverObstructionAllowPGNet (dbCoverObstructionId coverId,  
                                 Boolean          allowPGNet);
```

Description

Sets whether power and ground nets can be routed over the areas defined by this cover obstruction object.

dbIsCoverObstructionId

Boolean

```
dbIsCoverObstructionId(dbId objId);
```

Description

Return TRUE if the `objId` indicates a cover obstruction object.

dbPRBoundaryHasCoverObstruction

Boolean

```
dbPRBoundaryHasCoverObstruction(dbPRBoundaryIdboundaryId);
```

Description

Return TRUE if the specified PR boundary object has a cover obstruction object associated with it.

dbGetPRBoundaryCoverObstruction

dbCoverObstructionId

```
dbGetPRBoundaryCoverObstruction(dbPRBoundaryIdboundaryId);
```

Description

Gets the cover obstruction object associated with the specified PR boundary object. NULL is returned if the PR boundary has no cover obstruction.

Layer Halos

This section describes functions that affect layer halo objects.

About Layer Halo Objects

A layer halo is a blockage. It represents an extended area surrounding an instance or a PR boundary, which prevents objects on a specific layer from being allowed in the layer halo area. A layer halo extends the area surrounding the instance or PR boundary by specifying offset in 4 orthogonal directions.

There are two major difference between a layer halo and an area halo:

- An area halo is always of “placement” type. A layer halo is never of type “placement”.
- A layer halo blocks objects on a specific layer

Functions that are common to blockages also apply to a layer halo. These function include:

- [dbDeleteBlockage](#) on page 217
- [dbGetBlockage](#) on page 229
- [dbGetBlockageType](#) on page 222
- [dbGetBlockageOwner](#) on page 227

The following generators generate area halos along with other types of blockages.

- [dbStartGenBlockage](#) on page 242
- [dbStartGenBlockageToPoint](#) on page 244
- [dbStartGenClusterToBlockage](#) on page 253
- [dbGenClusterToBlockage](#) on page 254
- [dbStartGenInstToBlockage](#) on page 255
- [dbGenInstToBlockage](#) on page 256
- [dbStartGenPRBoundaryToBlockage](#) on page 257
- [dbGenPRBoundaryToBlockage](#) on page 258

Object Type

```
typedef dbId dbLayerHaloId;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

The following list the types of layer halos.

```
typedef enum dbBlockageType {
    dbcUnknownBlockageType = 0,
    dbcRoutingBlockage = 1,
    dbcViaRoutingBlockage = 2,
    dbcPlacementBlockage = 3,
    dbcWiringBlockage = 4,
    dbcFillBlockage = 5,
    dbcSlotBlockage = 6,
    dbcPinBlockage = 7,
    dbcFeedthruBlockage = 8,
    dbcScreenBlockage = 9,
} dbBlockageType;

#define dbcMaxBlockageType 9

const String dbvBlockageTypeString[] = {
    "unknown",
    "routing",
    "viaRouting",
    "placement",
    "wiring",
    "fill",
    "slot",
    "pin",
    "feedthru",
    "screen"
};
```


dbIsLayerHalold

```
dbLayerHaloId  
  
dbIsLayerHaloId(  
    dbId    objectId  
);
```

Description

This functions checks if the given Id is a valid dbLayerHalold.

Arguments

<i>objectId</i>	The object Id.

dbCreateLayerHalo

dbLayerHaloId

```
dbCreateLayerHalo (
    dbId          ownerId,
    dbLayer       layerNum,
    dbBlockageType type,
    dbDistance    topOffset,
    dbDistance    bottomOffset,
    dbDistance    leftOffset,
    dbDistance    rightOffset
);
```

Description

This function creates a layerHalo with the given attributes. The owner is validated and arguments are checked:

Arguments

<i>ownerId</i>	The owner of the layerHalo. Can only be an instance or a PR boundary. A warning will be generated if the owner type is wrong. This is enforced by OpenAccess.
<i>type</i>	The type of the layerHalo. The type cannot be “placement”. A warning will be generated if the type is wrong. This is also enforced by OpenAccess.
<i>layNum</i>	The layer to assign to the layerHalo.
<i>topOffset</i>	The top offset of the layerHalo.
<i>bottomOffset</i>	The bottom offset of the layerHalo.
<i>leftOffset</i>	The left offset of the layerHalo.
<i>rightOffset</i>	The right offset of the layerHalo.

dbGetLayerHaloLayerNum

dbLayer

```
dbGetLayerHaloLayerNum(  
    dbLayerHaloId    layerHaloId  
);
```

Description

This function returns the layer number of the given layerHalo.

Arguments

<i>layerHaloId</i>	The Id of the layerHalo.

dbSetLayerHaloLayerNum

```
dbSetLayerHaloLayerNum(  
    dbLayerHaloId    layerHaloId,  
    dbLayer          layerNum  
);
```

Description

This function moves the layerHalo to the specified layer.

Arguments

<i>layerHaloId</i>	The Id of the layerHalo.
<i>layerNum</i>	The layer to assign to the layerHalo.

dbGetLayerHaloLayerHeader

```
dbLayerHeaderId  
dbGetLayerHaloLayerHeader(  
    dbLayerHaloId    layerHaloId  
);
```

Description

This function returns the layer header for the layer of the given layerHalo.

Arguments

<i>layerHaloId</i>	The Id of the layerHalo.

dbGetLayerHaloOffsets

Boolean

```
dbGetLayerHaloOffsets (
    dbLayerHaloId    layerHaloId,
    dbDistance        *topOffset,
    dbDistance        *bottomOffset,
    dbDistance        *leftOffset,
    dbDistance        *rightOffset
);
```

Description

This function returns the offsets of the given layerHalo. The layerHalo is validated and the 4 offsets pointers are filled.

Arguments

<i>layerHaloId</i>	The Id of the layerHalo.
<i>topOffset</i>	The top offset of the layerHalo.
<i>bottomOffset</i>	The bottom offset of the layerHalo.
<i>leftOffset</i>	The left offset of the layerHalo.
<i>rightOffset</i>	The right offset of the layerHalo.

dbSetLayerHaloOffsets

Boolean

```
dbSetLayerHaloOffsets (
    dbLayerHaloId    layerHaloId,
    dbDistance        topOffset,
    dbDistance        bottomOffset,
    dbDistance        leftOffset,
    dbDistance        rightOffset
);
```

Description

This function sets the offsets of the given layerHalo. The layerHalo is validated and the offsets are set.

Arguments

<i>layerHaloId</i>	The Id of the layerHalo.
<i>topOffset</i>	The top offset of the layerHalo.
<i>bottomOffset</i>	The bottom offset of the layerHalo.
<i>leftOffset</i>	The left offset of the layerHalo.
<i>rightOffset</i>	The right offset of the layerHalo.

Boundaries

This section describes functions that affect boundary objects.

About Boundaries

A boundary object is used to define an enclosed area with a polygonal outline. Each segment of a boundary can have a name associated with it. A boundary is a figure, but not a shape.

There are 4 types of boundaries, area boundary, PR (place and route) boundary, snap boundary, and cluster boundary.

An area boundary defines an area in a cellview for a specific purpose. Area boundaries are named. A cellview can have multiple area boundaries, each boundary is identified by its name.

A PR boundary can have a coreBoxSpec and an IO box. CoreBoxSpec and IO box are special attributes. A coreBoxSpec defines a rectangular grid of sites (a row of rows) within a PR boundary that are available for cell placement. An IO box, together with the outline of the core box, defines the area for IO cell placement.

A PR boundary can own blockages. Area halos are the most common blockages owned by a PR boundary.

The Boundary Id is the following.

```
typedef dbId dbAreaBoundaryId;
```

The validation function verifies that the dbAreaBoundaryId is a valid oaAreaBoundary object.

```
inline oaAreaBoundary *  
dbiValidateAreaBoundary (dbAreaBoundaryId areaBoundaryId);
```


dbIsBoundaryId

Boolean

```
dbIsBoundaryId(  
    dbId      boundaryId  
);
```

Description

Verifies that the given `boundaryId` refers to a boundary object.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary object to be verified.

Return Values

Boolean	Returns TRUE if the <i>boundaryId</i> refers to a boundary object. Returns FALSE if it is not a boundary object.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsAreaBoundaryId

Boolean

```
dbIsAreaBoundaryId(  
    dbId      objId  
);
```

Description

Verifies that the given `objId` refers to an area boundary object.

Arguments

<i>objId</i>	The database ID for the object to be verified.

Return Values

Boolean	Returns TRUE if the ID refers to an area boundary object. Returns FALSE otherwise.

dbIsClusterBoundaryId

Boolean

```
dbIsClusterBoundaryId(  
    dbId      boundaryId  
);
```

Description

Verifies that the given `boundaryId` refers to a cluster boundary object.

Arguments

<i>boundaryId</i>	The database <code>dbboundaryId</code> for the cluster boundary object to be verified.

Return Values

Boolean	Returns TRUE if the <i>boundaryId</i> refers to a cluster boundary object. Returns FALSE if it is not a cluster boundary object.

dbIsPRBoundaryId

Boolean

```
dbIsPRBoundaryId(  
    dbId      boundaryId  
);
```

Description

Verifies that the given `boundaryId` refers to a PR boundary object.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the PR boundary object to be verified.

Return Values

Boolean	Returns TRUE if the <i>boundaryId</i> refers to a PR boundary object. Returns FALSE if it is not a PR boundary object.

dbIsSnapBoundaryId

Boolean

```
dbIsSnapBoundaryId(  
    dbId      boundaryId  
);
```

Description

Verifies that the given `boundaryId` refers to a snap boundary object.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the snap boundary object to be verified.

Return Values

Boolean	Returns TRUE if the <i>boundaryId</i> refers to a snap boundary object. Returns FALSE if it is not a snap boundary object.

dbCreateAreaBoundary

```
dbAreaBoundaryId  
dbCreateAreaBoundary(  
    dbCellViewId    cvId,  
    const char       *name,  
    unsigned int     nPoints,  
    const dbPoint    *pointArray,  
    const char       **edgeNames  
);
```

Description

Creates a new area boundary object in the cellview.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview.
<i>*name</i>	Name of the area boundary. Optional. If not specified, the database assigns using the format <code>B__n</code> , where <i>n</i> is a number that makes the boundary name unique.
<i>nPoints</i>	Specifies the number of points for the boundary.
<i>*pointArray</i>	Pointer to an array of points for the boundary. A point array with fewer than 3 points is an error. A NULL is returned and a warning issued. If the point array contains co-linear points, the array is compressed and the extra points removed.
<i>**edgeNames</i>	Contains the name of each edge. Optional. If specified, the number of names is assumed to be the same as the number of edges. If not specified, the database generates a name for each edge using the format <code>EDGE__n</code> , where <i>n</i> is a number starting at 0 and incrementing by 1 for each edge.

Return Values

<code>dbAreaBoundaryId</code>	Returns the database ID of the area boundary created. Returns a NULL ID, if the area boundary creation fails.

dbCreateClusterBoundary

```
dbClusterBoundaryId  
dbCreateClusterBoundary(  
    dbClusterId      clusterId,  
    const char       *name,  
    unsigned int      nPoints,  
    dbPoint           *pointArray,  
    const char       **edgeNames  
);
```

Description

Creates a new cluster boundary in the specified cluster with the specified attributes.

Arguments

<i>clusterId</i>	The database <code>dbClusterId</code> for the cluster.
<i>*name</i>	The name of the boundary.
<i>nPoints</i>	Specifies the number of points for the boundary.
<i>*pointArray</i>	Pointer to an array of points for the boundary.
<i>**edgeNames</i>	Contains the name of each edge. If given, the number of names is assumed to be the same as the number of edges. Otherwise, a name for each edge is automatically generated. Names take the form of <code>EDGE_n</code> , where <code>n</code> is an integer starting from 0 and incrementing by 1. Names are optional.

Return Values

<code>dbClusterBoundaryId</code>	Returns the database <code>dbClusterBoundaryId</code> for the cluster boundary created.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreatePRBoundary

```
dbPRBoundaryId
dbCreatePRBoundary(
    dbCellViewId    cvId,
    unsigned int     nPoints,
    dbPoint          *pointArray,
    const char       **edgeNames
);
```

Description

Creates a new PR boundary in the specified cell view with the specified attributes.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview.
<i>nPoints</i>	Specifies the number of points for the boundary.
<i>*pointArray</i>	Pointer to an array of points for the boundary.
<i>**edgeNames</i>	Contains the name of each edge. If given, the number of names is assumed to be the same as the number of edges. Otherwise, a name for each edge is automatically generated. Names take the form of <code>EDGE_n</code> , where <code>n</code> is an integer starting from 0 and incrementing by 1. Names are optional.

Return Values

<code>dbPRBoundaryId</code>	Returns the database <code>dbPRBoundaryId</code> for the PR boundary created.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateSnapBoundary

```
dbSnapBoundaryId
dbCreateSnapBoundary(
    dbCellViewId    cvId,
    unsigned int     nPoints,
    dbPoint          *pointArray,
    const char       **edgeNames
);
```

Description

Creates a new snap boundary in the specified cell view with the specified attributes.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cell view.
<i>nPoints</i>	Specifies the number of points for the boundary.
<i>*pointArray</i>	Points to an array of points for the boundary.
<i>**edgeNames</i>	A string array containing the name of each edge. If given, the number of names is assumed to be the same as the number of edges. Otherwise, a name for each edge is automatically generated. Names take the form of <code>EDGE_n</code> , where <code>n</code> is an integer starting from 0 and incrementing by 1. Names are optional.

Return Values

<code>dbSnapBoundaryId</code>	Returns the database <code>dbSnapBoundaryId</code> for the snap boundary created.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteBoundary

Boolean

```
dbDeleteBoundary(  
    dbBoundaryId    boundaryId  
);
```

Description

Deletes the boundary object specified by `boundaryId`.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary object to be deleted.

Return Values

Boolean	Returns TRUE if the boundary object was successfully deleted. Returns FALSE otherwise.

dbFindAreaBoundaryByName

```
dbAreaBoundaryId  
dbFindAreaBoundaryByName (  
    dbCellViewId      cvId,  
    const char        *name  
);
```

Description

Finds the area boundary with the given name in the cellview.

Arguments

<i>cvId</i>	The cellview to search for the area boundary.
<i>*name</i>	A text string containing the area boundary name.

Return Values

<i>dbAreaBoundaryId</i>	Returns the ID of the area boundary.

dbFindClusterBoundaryByName

```
dbClusterBoundaryId  
dbFindClusterBoundaryByName(  
    dbClusterId      clusterId,  
    const char       *name  
);
```

Description

Finds the cluster boundary with the given name in the cellview of the given cluster.

Arguments

<i>clusterId</i>	The database <code>dbClusterId</code> to search for a boundary by name.
<i>*name</i>	A text string containing the boundary name.

Return Values

<code>dbBoundaryId</code>	Returns a <code>dbClusterBoundaryId</code> found in the specified cluster.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetAreaBoundary

String

```
dbGetAreaBoundaryName (  
    dbAreaBoundaryId    boundaryId  
);
```

Description

Gets the name of the area boundary.

Arguments

<i>boundaryId</i>	The database ID of the area boundary.

Return Values

String	Returns the name of the area boundary object.

dbSetAreaBoundary

Boolean

```
dbSetAreaBoundaryName (
    dbAreaBoundaryId    boundaryId,
    const char          *name
);
```

Description

Sets the name of the area boundary.

Arguments

<i>boundaryId</i>	The database ID of the area boundary.
<i>*name</i>	Pointer to the area boundary name.

Return Values

Boolean	Returns TRUE if the area boundary name is set. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetPRBoundary

```
dbPRBoundaryId  
dbGetPRBoundary(  
    dbCellViewId    cvId  
);
```

Description

Gets the place and route (PR) boundary in the given cellview.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview to be searched for a PR boundary.

Return Values

<code>dbBoundaryId</code>	Returns the <code>dbPRBoundaryId</code> for the PR boundary object found.

dbGetPRBoundaryCoreBoxSpec

Boolean

```
dbGetPRBoundaryCoreBoxSpec (
    dbPRBoundaryId    prBoundaryId,
    dbCoreBoxSpec      *coreBoxSpec
);
```

Description

Gets the coreBoxSpec of the PR boundary.

Arguments

<i>prBoundaryId</i>	The PR boundary ID.
<i>*coreBoxSpec</i>	Pointer to the coreBoxSpec.

Return Values

Boolean	Returns TRUE if the function successfully gets the coreBoxSpec. Returns FALSE otherwise.

dbSetPRBoundaryCoreBoxSpec

Boolean

```
dbGetPRBoundaryCoreBoxSpec (
    dbPRBoundaryId      prBoundaryId,
    const dbCoreBoxSpec *coreBoxSpec
);
```

Description

Sets the coreBoxSpec of the PR boundary.

Arguments

<i>prBoundaryId</i>	The PR boundary ID.
<i>*coreBoxSpec</i>	Pointer to the coreBoxSpec.

Return Values

Boolean	Returns TRUE if the function successfully sets the coreBoxSpec. Returns FALSE otherwise.

dbRemovePRBoundaryCoreBoxSpec

Boolean

```
dbRemovePRBoundaryCoreBoxSpec (  
    dbPRBoundaryId      prBoundaryId  
);
```

Description

Removes the coreBoxSpec of the PR boundary.

Arguments

<i>prBoundaryId</i>	The PR boundary ID.

Return Values

Boolean	Returns TRUE if the function successfully removes the coreBoxSpec. Returns FALSE otherwise.

dbGetPRBoundaryIOBox

```
dbBBox *  
dbGetPRBoundaryIOBoxSpec (  
    dbPRBoundaryId    prBoundaryId  
);
```

Description

Gets the IO box of the PR boundary.

Arguments

<i>prBoundaryId</i>	The PR boundary ID.

Return Values

dbBBox *	Returns the IO box of the PR boundary.

dbSetPRBoundaryIOBox

Boolean

```
dbSetPRBoundaryIOBox(  
    dbPRBoundaryId    prBoundaryId,  
    const dbBBox      *bBox  
);
```

Description

Sets the IO box of the PR boundary.

Arguments

<i>prBoundaryId</i>	The PR boundary ID.
<i>*bBox</i>	Pointer to the IO box.

Return Values

Boolean	Returns TRUE if the function successfully sets the IO box. Returns FALSE otherwise.

dbRemovePRBoundaryIOBox

Boolean

```
dbRemovePRBoundaryIOBox(  
    dbPRBoundaryId    prBoundaryId  
);
```

Description

Removes the IO box of the PR boundary.

Arguments

<i>prBoundaryId</i>	The PR boundary ID.

Return Values

Boolean	Returns TRUE if the function successfully removes the IO box. Returns FALSE otherwise.

dbStartGenPRBoundaryToBlockage

```
dbGenStateId  
dbStartGenPRBoundaryToBlockage (  
    dbPRBoundaryId    prBoundaryId  
);
```

Description

Starts generating all blockages owned by the PR boundary.

Arguments

<i>prBoundaryId</i>	The database ID of the PR boundary.

Return Values

<i>dbGenStateId</i>	Returns the generator <i>dbGenStateId</i> used to track the state of the blockage generation.

dbGetCoreBoxSpecNumRows

```
int  
dbGetCoreBoxSpecNumRows (  
    const dbCoreBoxSpec      *coreBoxSpec  
);
```

Description

Gets the number of rows in the PR boundary coreBoxSpec.

Arguments

<i>*coreBoxSpec</i>	Pointer to the coreBoxSpec.

Return Values

<i>int</i>	Returns the number of rows in the coreBoxSpec.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetCoreBoxSpecRowBBox

```
dbBBox *  
dbGetCoreBoxSpecNumRows (  
    const dbCoreBoxSpec    *coreBoxSpec,  
    int                    rowNum  
);
```

Description

Gets the bounding box of the row indexed by the specified number in the PR boundary coreBoxSpec.

Arguments

<i>*coreBoxSpec</i>	Pointer to the coreBoxSpec.
<i>int</i>	The row number index.

Return Values

dbBBox *	Returns the bounding box of the row indexed.

dbGetSnapBoundary

```
dbSnapBoundaryId  
dbGetSnapBoundary(  
    dbCellViewId    cvId  
);
```

Description

Gets the snap boundary in the given cellview.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview to be searched for a snap boundary.

Return Values

<code>dbBoundaryId</code>	Returns the <code>dbSnapBoundaryId</code> for the snap boundary object found.

dbTransformBoundary

Boolean

```
dbTransformBoundary(  
    dbBoundaryId    boundaryId,  
    float           scale,  
    float           angle  
);
```

Description

Transforms the boundary object specified by `boundaryId` according to the specified scale factor and rotation angle.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary to be transformed.
<i>scale</i>	Scale to which to magnify.
<i>angle</i>	Angle at which to rotate in radians.

Return Values

Boolean	Returns TRUE if the boundary object was successfully transformed. Returns FALSE otherwise.

dbGetClusterBoundaryName

String

```
dbGetClusterBoundaryName(  
    dbClusterBoundaryId    boundaryId,  
    );
```

Description

Returns the name for the given cluster boundary.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> to search for a cluster boundary name.

Return Values

String	Returns a text string containing the name of the specified cluster boundary.

dbSetClusterBoundaryName

Boolean

```
dbSetClusterBoundaryName (
    dbClusterBoundaryId    boundaryId,
    const char              *name
);
```

Description

Sets the name of the given cluster boundary to the specified name.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary object.
<i>*name</i>	The name to be given to the cluster boundary.

Return Values

Boolean	Returns TRUE if the name for the cluster boundary was successfully set. Returns FALSE otherwise.

dbGetClusterBoundaryCluster

```
dbClusterId  
dbGetClusterBoundaryCluster(  
    dbClusterBoundaryId    boundaryId,  
    );
```

Description

Gets the cluster for the given cluster boundary.

Arguments

<i>boundaryId</i>	The database <code>dbClusterBoundaryId</code> for the boundary object.

Return Values

<code>dbClusterId</code>	Returns the <code>clusterId</code> of the specified cluster boundary.

dbIsBoundaryOrthogonal

Boolean

```
dbIsBoundaryOrthogonal(  
    dbBoundaryId    boundaryId  
);
```

Description

Checks that the point array for a boundary is orthogonal.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary object.

Return Values

Boolean	Returns TRUE if the point array for the boundary is orthogonal. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsBoundaryRectangle

Boolean

```
dbIsBoundaryRectangle(  
    dbBoundaryId    boundaryId  
);
```

Description

Checks that a boundary is a rectangle.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary object.

Return Values

Boolean	Returns TRUE if the boundary is a Rectangle. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetBoundaryArea

```
double  
dbGetBoundaryArea(  
    dbBoundaryId    boundaryId  
);
```

Description

Gets the area in database units (DBU) squared the boundary specified by the database BoundaryId.

Arguments

<i>boundaryId</i>	The database dbBoundaryId for the boundary object.

Return Values

double	Returns the area in DBU squared for the boundary specified by BoundaryId.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetBoundaryEdge

Boolean

```
dbGetBoundaryEdge (
    dbBoundaryId    boundaryId,
    unsigned int    nPoints,
    dbPoint         *pointArray,
    string          **edgeNames
);
```

Description

Gets the number, the pointArray, and the name for each edge of the boundary given in *boundaryId*.

Arguments

<i>boundaryId</i>	The database <i>dbBoundaryId</i> for the boundary object.
<i>nPoints</i>	Pointer to integer to be filled with the number of points for the boundary.
<i>*pointArray</i>	Pointer to <i>dbPoint</i> array to be filled with the points for the boundary.
<i>**edgeNames</i>	Pointer to String array to be filled with the names for the edges of the boundary.

Return Values

Boolean	Returns TRUE if the edges for <i>boundaryId</i> were successfully retrieved. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetBoundaryEdge

Boolean

```
dbSetBoundaryEdge (
    dbBoundaryId    boundaryId,
    unsigned int     nPoints,
    dbPoint          *pointArray,
    string           **edgeNames
);
```

Description

Sets the edges of the boundary given in `boundaryId` to the given edges. The `edgeNames` attribute is optional.

Arguments

<i>boundaryId</i>	The database <code>dbBoundaryId</code> for the boundary object.
<i>nPoints</i>	Pointer to integer to be filled with the number of points for the boundary.
<i>*pointArray</i>	Pointer to <code>dbPoint</code> array to be filled with the points for the boundary.
<i>**edgeNames</i>	A string array containing the name of each edge. If given, the number of names is assumed to be the same as the number of edges. Otherwise, a name for each edge is automatically generated. Names take the form of <code>EDGE_n</code> , where <code>n</code> is an integer starting from 0 and incrementing by 1. Names are optional.

Return Values

Boolean	Returns TRUE if the edges for <i>boundaryId</i> were successfully set. Returns FALSE otherwise.

dbStartGenAreaBoundary

```
dbGenStateId  
dbStartGenAreaBoundary(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the area boundaries in the cellview.

The function returns only area boundaries, no other boundary types.

Note: A cellview can contain only one PR boundary and one snap boundary. This function does not return either. Clusters own cluster boundaries. To generate cluster boundaries, first generate clusters in the cellview, then generate cluster boundaries for each cluster.

Arguments

<i>cvId</i>	The database ID of the cellview.

Return Values

<i>dbGenStateId</i>	Returns the generator <i>dbGenStateId</i> used to track the state of area boundary generation.

dbGenAreaBoundary

Boolean

```
dbGenAreaBoundary(  
    dbGenStateId    pState,  
    dbAreaBoundaryId *areaBoundaryId  
);
```

Description

Generates all the area boundaries in the cellview.

Arguments

<i>pState</i>	The generator dbGenStateId used to track the state of area boundary generation.
<i>*areaBoundaryId</i>	The dbAreaBoundaryId pointer, filled if an area boundary is generated.

Return Values

Boolean	Returns TRUE if an area boundary is generated. Returns FALSE otherwise.

dbStartGenClusterToBoundary

```
dbGenStateId  
dbStartGenClusterToBoundary(  
    dbClusterId    clusterId  
);
```

Description

Starts generating all the boundaries in a cluster.

Arguments

<i>clusterId</i>	The database <code>dbClusterId</code> for the cluster object.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> used to track the state of boundary generation.

dbGenClusterToBoundary

Boolean

```
dbGenClusterToBoundary(  
    dbGenStateId          pState,  
    dbClusterBoundaryId   *boundaryId  
);
```

Description

Generates the next boundary in the cluster.

Arguments

<i>pState</i>	The generator dbGenStateId used to track the state of boundary generation.
<i>*boundaryId</i>	The pointer to the dbClusterBoundaryId. Returned if a boundary is generated.

Return Values

Boolean	Returns TRUE if a boundary is generated. Returns FALSE otherwise.

dbStartGenBoundaryToPoint

```
dbGenStateId  
dbStartGenBoundaryToPoint(  
    dbBoundaryId      BoundaryId  
);
```

Description

Starts generating all the points for the given boundary.

Arguments

<i>boundaryId</i>	The database <code>dbMarkerId</code> for the boundary object.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> used to track the state of boundary generation.

dbGenBoundaryToPoint

Boolean

```
dbGenBoundayToPoint(  
    dbGenStateId    pState,  
    dbPoint         *point  
);
```

Description

Generates the next point in the boundary.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of point generation.
<i>*point</i>	A pointer to <code>dbPoint</code> . Returned if a point is generated.

Return Values

Boolean	Returns TRUE if a point is generated. Returns FALSE otherwise.

Clusters

The following functions generate clusters within cellViews.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenCluster

```
dbGenStateId  
dbStartGenCluster(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the cluster objects in the specified cellview.

Arguments

<i>cvId</i>	The cellview ID in which to generate the cluster objects.

Return Values

<i>dbGenStateId</i>	Returns the generator <i>dbGenStateId</i> used to track the state of cluster generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenCluster

Boolean

```
dbGenCluster(  
    dbGenStateId    pState,  
    dbCluster*      clusterId  
);
```

Description

Generates the next cluster in the cellview.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of cluster generation.
<i>clusterId</i>	The <code>dbCluster</code> pointer. Returned by the function if a cluster is generated.

Return Values

Boolean	Returns TRUE if a cluster is generated. Returns FALSE otherwise.

LayerHeaders

A layerHeader indicates whether specific objects are using a layer in the design hierarchy. The objects can be steiners, guides, blockages, or lppHeaders. The layerHeader provides direct access to the objects grouped by layer.

The OpenAccess database automatically manages layerHeaders.

The LayerHeader Id is the following.

```
typedef dbId dbLayerHeadid;
```

The validation function verifies that the specified dbLayerHeaderId maps to a valid oaLayerHeader object.

```
inline oaLayerHeader *  
dbiValidateLayerHeader (dbLayerHeaderId layerHeaderId);
```

dbIsLayerHeaderId

Boolean

```
dbIsLayerHeaderId(  
    dbId      id  
);
```

Description

Checks that the *id* refers to a valid *dbLayerHeaderId*.

Arguments

<i>id</i>	The database identifier of the layer header.

Return Values

Boolean	Returns TRUE if the <i>id</i> refers to a layer header. Returns FALSE otherwise.

dbFindLayerHeader

```
dbLayerHeaderId  
dbFindLayerHeader(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Finds the layer header on the specified layer in the specified cellview.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>layerNum</i>	The layer number on which to find the layer header.

Return Values

<i>dbLayerHeaderId</i>	Returns the identifier of the layer header.

dbGetLayerHeaderLayerNum

```
dbLayer  
dbGetLayerHeaderLayerNum(  
    dbLayerHeaderId    layerHeaderId  
);
```

Description

Gets the layer number of the layer header.

Arguments

<i>layerHeaderId</i>	The database identifier of the layer header.

Return Values

<i>dbLayer</i>	Returns the layer number of the specified layer header.

dbIsLayerHeaderBound

Boolean

```
dbIsLayerHeaderBound(  
    dbLayerHeaderId    layerHeaderId  
);
```

Description

Checks whether the layer header is bound to a technology database layer.

A layer header can be bound or unbound. A bound layer header has a valid technology database object. An unbound layer header has a layer number only. Layer header binding occurs when a layer header is read in or a new layer header is created.

Arguments

<i>layerHeaderId</i>	The database identifier of the layer header.

Return Values

Boolean	Returns TRUE if the layer header is bound to a technology database layer. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetLayerSteinerBBox

```
dbBBox *  
dbGetLayerSteinerBBox(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Gets the combined BBox of the steiners from the layer header of the layer.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>layerNum</i>	The layer number. The function uses the layer number to obtain the layer header in the cellview.

Return Values

<i>dbBBox *</i>	Returns the combined BBox of the steiners from the layer header of the layer. Returns FALSE or NULL if no layer header is found.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetLayerGuideBBox

```
dbBBox *  
dbGetLayerGuideBBox(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Gets the combined BBox of the guides from the layer header of the layer.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>layerNum</i>	The layer number. The function uses the layer number to obtain the layer header in the cellview.

Return Values

<i>dbBBox *</i>	Returns the combined BBox of the guides from the layer header of the layer. Returns FALSE or NULL if no layer header is found.

dbGetLayerBlockageBBox

```
dbBBox *  
dbGetLayerBlockageBBox(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Gets the combined BBox of the blockages from the layer header of the layer.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>layerNum</i>	The layer number. The function uses the layer number to obtain the layer header in the cellview.

Return Values

<i>dbBBox *</i>	Returns the combined BBox of the blockages from the layer header of the layer. Returns FALSE or NULL if no layer header is found.

dbGetLPPHeaderLayerHeader

```
dbLayerHeaderId  
dbGetLPPHeaderLayerHeader (  
    dbLPPHeaderId      lppHeaderId  
);
```

Description

Gets the layerHeader with which the lppHeader associates.

Arguments

<i>lppHeaderId</i>	The database dbLPPHeaderId of the layer header.

Return Values

<i>dbLayerHeaderId</i>	Returns the database identifier of the layerHeader with which the lppHeader associates.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenLayerHeader

```
dbGenStateId  
dbStartGenLayerHeader(  
    dbCellViewId    cvId  
);
```

Description

Starts layerHeader generation in the specified cellview.

Arguments

<i>cvId</i>	The database dbCellViewId of the cellview.

Return Values

dbGenStateId	Returns the generator dbGenStateId which keeps track of the state of layerHeader generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenLayerHeader

Boolean

```
dbGenLayerHeader(  
    dbGenStateId      pState,  
    dbLayerHeaderId   *layerHeaderId  
);
```

Description

Generates the layer headers in the cellview.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of layer header generation.
<i>*layerHeaderId</i>	A pointer to the <code>dbLayerHeaderId</code> . Returned if a layer header is generated.

Return Values

Boolean	Returns TRUE if a layer header is generated. Returns FALSE otherwise.

dbStartGenLayerToSteiner

```
dbGenStateId  
dbStartGenLayerToSteiner(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Starts steiner generation on the layer header of the specified layer.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> of the cellview.
<i>layerNum</i>	The layer number. The function uses the layer number to obtain the layer header in the cellview.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> which keeps track of the state of steiner generation.

dbGenLayerToSteiner

Boolean

```
dbGenLayerToSteiner(  
    dbGenStateId    pState,  
    dbSteinerId     *steinerId  
);
```

Description

Generates the steiners on the layer header of the specified layer.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of steiner generation.
<i>*steinerId</i>	The <code>dbSteinerId</code> pointer. Returned if a steiner is generated.

Return Values

Boolean	Returns TRUE if a steiner is generated. Returns FALSE otherwise.

dbStartGenLayerToGuide

```
dbGenStateId  
dbStartGenLayerToGuide(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Starts guide generation on the layer header of the specified layer.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> of the cellview.
<i>layerNum</i>	The layer number. The function uses the layer number to obtain the layer header in the cellview.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> which keeps track of the state of guide generation.

dbGenLayerToGuide

Boolean

```
dbGenLayerToGuide(  
    dbGenStateId    pState,  
    dbGuideId       *guideId  
);
```

Description

Generates the guides on the layer header of the specified layer.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of guide generation.
<i>*guideId</i>	The <code>dbSteinerId</code> pointer. Returned if a guide is generated.

Return Values

Boolean	Returns TRUE if a guide is generated. Returns FALSE otherwise.

dbStartGenLayerToBlockage

```
dbGenStateId  
dbStartGenLayerToBlockage(  
    dbCellViewId    cvId,  
    dbLayer          layerNum  
);
```

Description

Starts blockage generation on the layer header of the specified layer.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> of the cellview.
<i>layerNum</i>	The layer number. The function uses the layer number to obtain the layer header in the cellview.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> which keeps track of the state of blockage generation.

dbGenLayerToBlockage

Boolean

```
dbGenLayerToBlockage (
    dbGenStateId    pState,
    dbBlockageId    *blockageId
);
```

Description

Generates the blockages on the layer header of the specified layer.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of blockage generation.
<i>*blockageId</i>	The <code>dbSteinerId</code> pointer. Returned if a blockage is generated.

Return Values

Boolean	Returns TRUE if a blockage is generated. Returns FALSE otherwise.

dbStartGenLayerHeaderToLPPHeader

```
dbGenStateId  
dbStartGenLayerHeaderToLPPHeader (  
    dbLayerHeaderId    layerHeaderId);
```

Description

Starts generating all the lppHeaders that contain the layer of the specified layerHeader.

Arguments

<i>layerHeaderId</i>	The database dbLayerHeaderId of the cellview.

Return Values

dbGenStateId	Returns the generator dbGenStateId which keeps track of the state of lppHeader generation.

dbGenLayerHeaderToLPPHeader

Boolean

```
dbGenLayerHeaderToLPPHeader (  
    dbGenStateId      pState,  
    dbLPPHeaderId     *lppHeaderId  
);
```

Description

Generates the lppHeaders that contain the layer of the specified layerHeader.

Arguments

<i>pState</i>	The generator dbGenStateId which keeps track of the state of lppHeader generation.
<i>*lppHeaderId</i>	The dbSteinerId pointer. Returned if an lppHeader is generated.

Return Values

Boolean	Returns TRUE if an lppHeader is generated. Returns FALSE otherwise.

Markers

This section describes functions that affect marker objects.

About Markers

Markers are used to highlight a concept in a design, such as a design error, a warning condition, or any information generated by a tool. Typically, markers are used to report DRC violations, such as minimum spacing violation rules, shorts and antenna. The marker object itself stores the coordinates of the marker shape, but no layer information. Markers optionally store their relationship to the objects that caused them.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsMarkerId

Boolean

```
dbIsMarkerId(  
    dbMarkerId    markerId  
);
```

Description

Checks that the given `markerId` refers to a marker object.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker.

Return Values

Boolean	Returns TRUE if the <i>markerId</i> refers to a marker object. Returns FALSE if it is not a marker object.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateMarker

dbMarkerId

```
dbCreateMarker(  
    dbCellViewId      cvId,  
    const char         *msg,  
    const char         *shortMsg,  
    const char         *tool,  
    Boolean            isVisible,  
    Boolean            isClosed,  
    dbMarkerSeverity   severity,  
    dbDeleteWhenType   deleteWhen,  
    unsigned int        nPoints,  
    dbPoint            *pointArray,  
    unsigned int        nObjects,  
    dbId               *objectIds  
);
```

Description

Creates a marker object in the cellview `cvId` with the specified attributes.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview.
<i>*msg</i>	String of the message associated with the marker.
<i>*shortMsg</i>	String of the short message associated with the marker.
<i>*tool</i>	String of the tool associated with the marker.
<i>isVisible</i>	Boolean setting the <code>isVisible</code> field for the specified object.
<i>isClosed</i>	Boolean indicating the point array associated with the marker as closed.
<i>severity</i>	Sets the <i>severity</i> field for the specified marker. The valid enum choices are: <code>dbcMarkerAnnotation</code> <code>dbcMarkerInfo</code> <code>dbcMarkerAcknowledgedWarning</code> <code>dbcMarkerWarning</code> <code>dbcMarkerSignedOffError</code> <code>dbcMarkerError</code> <code>dbcMarkerSignedOffCriticalError</code> <code>dbcMarkerCriticalError</code> <code>dbcMarkerFatalError</code>
<i>deleteWhen</i>	The enum value indicates when the database automatically deletes the marker. The valid enum choices are: <code>dbcDeleteNever</code> <code>dbcDeleteFirst</code> <code>dbcDeleteLast</code> <code>dbcDeleteModify</code>
<i>nPoints</i>	The number of Points for this marker.
<i>*pointArray</i>	Pointer to an array that containing the points for the marker.
<i>nObjects</i>	The number of objects associated with the marker.
<i>*objectIds</i>	An array of object ids that are associated with the marker.

Return Values

<code>dbMarkerId</code>	Returns the database <code>dbMarkerId</code> for the marker if a marker object is created. Returns <code>FALSE</code> otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteMarker

Boolean

```
dbDeleteMarker(  
    dbMarkerId    markerId  
);
```

Description

Deletes the marker object specified by `markerId`.

Arguments

<code>markerId</code>	The database <code>dbMarkerId</code> for the marker to be deleted.

Return Values

Boolean	Returns TRUE if the marker object was successfully deleted. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbTransformMarker

Boolean

```
dbTransformMarker(  
    dbMarkerId    markerId,  
    float         scale,  
    float         angle  
);
```

Description

Transforms the marker object specified by `markerId` according to the specified scale factor and rotation angle.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be transformed.
<i>scale</i>	Scale to be magnified to.
<i>angle</i>	Angle to be rotated in radians.

Return Values

Boolean	Returns TRUE if the marker object was successfully transformed. Returns FALSE otherwise.

dbAddObjectToMarker

Boolean

```
dbAddObjectToMarker(  
    dbMarkerId    markerId,  
    dbId          objectId  
);
```

Description

Adds an object to the marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker.
<i>objectId</i>	The ID of the object to be added to the marker.

Return Values

Boolean	Returns TRUE if an object was successfully added to the marker. Returns FALSE otherwise.

dbRemoveObjectFromMarker

Boolean

```
dbRemoveObjectFromMarker(  
    dbMarkerId    markerId,  
    dbId          objectId  
);
```

Description

Removes an object from the marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker.
<i>objectId</i>	The ID of the object to be removed from the marker.

Return Values

Boolean	Returns TRUE if an object was successfully removed from the marker. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsMarkerVisible

Boolean

```
dbIsMarkerVisible(  
    dbMarkerId    markerId,  
    );
```

Description

Checks that the marker specified by the database `markerId` is visible.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

Boolean	Returns TRUE if the marker specified by <code>markerId</code> is visible. Returns FALSE otherwise.

dbSetMarkerVisible

Boolean

```
dbSetMarkerVisible(  
    dbMarkerId    markerId,  
    Boolean       isVisible  
);
```

Description

Sets the `isVisible` field for the specified marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be set.
<i>isVisible</i>	Sets the <code>isVisible</code> field for the specified marker.

Return Values

Boolean	Returns TRUE if the <code>isVisible</code> field for the specified marker is successfully set. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsMarkerClosed

Boolean

```
dbIsMarkerClosed(  
    dbMarkerId    markerId  
);
```

Description

Verifies that the marker specified by the database `markerId` is closed.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

Boolean	Returns TRUE if the marker specified by <code>markerId</code> is closed. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetMarkerClosed

Boolean

```
dbSetMarkerClosed(  
    dbMarkerId    markerId,  
    Boolean       isClosed  
);
```

Description

Sets the `isClosed` field for the specified marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be set.
<i>isClosed</i>	Sets the <code>isClosed</code> field for the specified marker.

Return Values

Boolean	Returns TRUE if the <code>isClosed</code> field of the specified marker is successfully set. Returns FALSE otherwise.

dbGetMarkerSeverity

```
dbMarkerSeverity  
dbGetMarkerSeverity(  
    dbMarkerId    markerId,  
    );
```

Description

Gets the severity of the violation shown by the given marker, defaults to info.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>dbMarkerSeverity</code>	Gets the <i>severity</i> field for the specified marker. The default is info. The valid enum choices are: The valid enum choices are: dbcMarkerAnnotation dbcMarkerInfo dbcMarkerAcknowledgedWarning dbcMarkerWarning dbcMarkerSignedOffError dbcMarkerError dbcMarkerSignedOffCriticalError dbcMarkerCriticalError dbcMarkerFatalError

dbSetMarkerSeverity

Boolean

```
dbSetMarkerSeverity(  
    dbMarkerId          markerId,  
    dbMarkerSeverity    severity  
);
```

Description

Sets the `severity` field for the specified marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be set.
<i>severity</i>	Sets the <code>severity</code> field for the specified marker. The valid enum choices are: The valid enum choices are: dbcMarkerAnnotation dbcMarkerInfo dbcMarkerAcknowledgedWarning dbcMarkerWarning dbcMarkerSignedOffError dbcMarkerError dbcMarkerSignedOffCriticalError dbcMarkerCriticalError dbcMarkerFatalError

Return Values

Boolean	Returns TRUE if the <code>severity</code> field of the specified marker is successfully set. Returns FALSE otherwise.

dbMarkerSeverityToString

```
String  
dbMarkerSeverityToString(  
    dbMarkerSeverity    severity  
);
```

Description

Converts an enumerated value to a string for the severity attribute of a marker object.

Arguments

<i>severity</i>	The enumerated value of the <i>severity</i> attribute of the specified marker object. The valid enum choices are: dbcMarkerAnnotation dbcMarkerInfo dbcMarkerAcknowledgedWarning dbcMarkerWarning dbcMarkerSignedOffError dbcMarkerError dbcMarkerSignedOffCriticalError dbcMarkerCriticalError dbcMarkerFatalError

Return Values

String	Returns the name of the severity attribute of a marker object. The valid string values are: "annotation" "info" "acknowledgedWarning" "warning" "signedOffError" "error" "signedOffCriticalError" "criticalError" "fatalError"

dbStringToMarkerSeverity

```
dbMarkerSeverity  
dbStringToMarkerSeverity(  
    constString    s  
);
```

Description

Converts a string to an enumerated value for the severity attribute of a marker object.

Arguments

<i>s</i>	The name of the <code>severity</code> attribute of the specified marker object. The valid string values are: "annotation" "info" "acknowledgedWarning" "warning" "signedOffError" "error" "signedOffCriticalError" "criticalError" "fatalError"

Return Values

<code>dbMarkerSeverity</code>	Returns the enumerated value of the severity attribute of a marker object. The valid enum values are: <code>dbcMarkerAnnotation</code> <code>dbcMarkerInfo</code> <code>dbcMarkerAcknowledgedWarning</code> <code>dbcMarkerWarning</code> <code>dbcMarkerSignedOffError</code> <code>dbcMarkerError</code> <code>dbcMarkerSignedOffCriticalError</code> <code>dbcMarkerCriticalError</code> <code>dbcMarkerFatalError</code>

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetMarkerMsg

```
String  
dbGetMarkerMsg(  
    dbMarkerId    markerId,  
    );
```

Description

Gets the message associated with the marker given in the database `markerId`

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>String</code>	Returns the message associated with the marker.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetMarkerMsg

Boolean

```
dbSetMarkerMsg(  
    dbMarkerId      markerId,  
    const char      *msg  
);
```

Description

Associates the message with the specified marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be set.
<i>*msg</i>	Message to be associated with the specified marker.

Return Values

Boolean	Returns TRUE if the message is successfully associated with the specified marker. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetMarkerShortMsg

```
String  
dbGetMarkerShortMsg(  
    dbMarkerId    markerId  
);
```

Description

Gets the short message associated with the marker given in the database `markerId`

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>String</code>	Returns the short message associated with the marker.

dbSetMarkerShortMsg

Boolean

```
dbSetMarkerShortMsg(  
    dbMarkerId      markerId,  
    const char      *shortMsg  
);
```

Description

Associates the short message with the specified marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be set.
<i>*shortMsg</i>	Short message to be associated with the specified marker.

Return Values

Boolean	Returns TRUE if the short message is successfully associated with the specified marker. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetMarkerTool

```
String  
dbGetMarkerTool(  
    dbMarkerId    markerId  
);
```

Description

Gets the tool name associated with the marker given in the database `markerId`

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>String</code>	Returns the tool name associated with the marker.

dbSetMarkerTool

Boolean

```
dbSetMarkerTool(  
    dbMarkerId      markerId,  
    const char      *tool  
);
```

Description

Sets the tool name Associated with the specified marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker to be set.
<i>*tool</i>	Tool name to be associated with the specified marker.

Return Values

Boolean	Returns TRUE if the tool name is successfully associated with the specified marker. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetMarkerDeleteWhen

```
dbDeleteWhenType  
dbGetMarkerDeleteWhen(  
    dbMarkerId    markerId  
);
```

Gets the `deleteWhen` attribute for the marker specified by `markerId`.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>dbDeleteWhenType</code>	Returns the value of the <code>deleteWhen</code> marker. The valid enum choices are: <code>dbcDeleteNever</code> <code>dbcDeleteFirst</code> <code>dbcDeleteLast</code> <code>dbcDeleteModify</code>

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetMarkerDeleteWhen

Boolean

```
dbSetMarkerDeleteWhen(  
    dbMarkerId      markerId,  
    dbDeleteWhenType deleteWhen  
);
```

Sets the `deleteWhen` attribute for the marker specified by `markerId`.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.
<i>deleteWhen</i>	The value of the <code>deleteWhen</code> attribute. The valid enum choices are: dbcDeleteNever dbcDeleteFirst dbcDeleteLast dbcDeleteModify

Return Values

Boolean	Returns TRUE if the <code>deleteWhen</code> attribute was successfully set. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetMarker

Boolean

```
dbGetMarker(  
    dbMarkerId      markerId,  
    unsigned int    *nPoints,  
    dbPoint         **pointArray  
);
```

Description

Gets the points for the marker specified by `markerId`.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.
<i>*nPoints</i>	A pointer to the number of marker points. Returned by the function if points are retrieved.
<i>**pointArray</i>	The pointer to the marker points array. Returned by the function if points are retrieved.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetMarkerPoint

Boolean

```
dbSetMarkerPoint(  
    dbMarkerId      markerId,  
    unsigned int    nPoints,  
    dbPoint         *pointArray  
);
```

Description

Sets the point array for the marker specified by `markerId`.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.
<i>nPoints</i>	The number of marker points to be set for the marker.
<i>*pointArray</i>	The points array to be set for the marker.

Return Values

Boolean	Returns TRUE if the <code>point</code> field of the specified marker is set. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenMarker

```
dbGenStateId  
dbStartGenMarker(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the markers for a cellview.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> used to track the state of marker generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenMarker

Boolean

```
dbGenMarker (
    dbGenStateId    pState,
    dbMarkerId      *pMarkerId
);
```

Description

Generates the next marker in the cellview.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of marker generation.
<i>*pMarkerId</i>	The <code>dbMarkerId</code> pointer. Returned by the function if a marker is generated.

Return Values

Boolean	Returns TRUE if a marker is generated. Returns FALSE otherwise.

dbStartGenMarkerToPoint

```
dbGenStateId  
dbStartGenMarkerToPoint(  
    dbMarkerId    markerId  
);
```

Description

Starts generating all the points for the given marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> used to track the state of point generation.

dbGenMarkerToPoint

Boolean

```
dbGenMarkerToPoint(  
    dbGenStateId    pState,  
    dbPoint         *point  
);
```

Description

Generates the next point in the marker.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of point generation.
<i>*point</i>	The <code>dbPoint</code> pointer. Returned by the function if a point is generated.

Return Values

Boolean	Returns TRUE if a point is generated. Returns FALSE otherwise.

dbStartGenMarkerToObject

```
dbGenStateId  
dbStartGenMarkerToObject (  
    dbMarkerId      markerId  
);
```

Description

Starts generating all the objects for the given marker.

Arguments

<i>markerId</i>	The database <code>dbMarkerId</code> for the marker object.

Return Values

<code>dbGenStateId</code>	Returns the generator <code>dbGenStateId</code> used to track the state of object generation.

dbGenMarkerToObject

Boolean

```
dbGenMarkerToObject (
    dbGenStateId    pState,
    dbId            *objId
);
```

Description

Generates the next object for the given marker.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of object generation.
<i>*objId</i>	The <code>objId</code> pointer. Returned by the function if an object is generated.

Return Values

Boolean	Returns TRUE if an object is generated. Returns FALSE otherwise.

dbStartGenObjectToMarker

```
dbGenStateId  
dbStartGenObjectToMarker(  
    dbId      objId  
);
```

Description

Starts generating all the markers for the given object.

Arguments

<i>objId</i>	The database <code>dbId</code> for the object.

Return Values

<code>dbGenStateId</code>	The generator <code>dbGenStateId</code> used to track the state of marker generation.

dbGenObjectToMarker

Boolean

```
dbGenObjectToMarker(  
    dbGenStateId    pState,  
    dbMarkerId      *markerId  
);
```

Description

Generates the next marker on the object.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of marker generation.
<i>*markerId</i>	The <code>dbMarkerId</code> pointer. Returned by the function if a marker is generated.

Return Values

Boolean	Returns TRUE if a marker is generated. Returns FALSE otherwise.

RowHeaders

This section describes functions that affect row headers.

About rowHeaders

A `rowHeader` contains information about a particular `siteDef` and a list of all the rows using that `siteDef`.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsRowHeaderId

Boolean

```
dbIsRowHeaderId(  
    dbRowHeaderId    rowHeaderId  
);
```

Description

Checks that the given `rowHeaderId` refers to a `rowHeader` object.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.

Return Values

Boolean	Returns TRUE if the <i>rowHeaderId</i> refers to a <code>rowHeader</code> object. Returns FALSE if it is not a row object.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindRowHeader

```
dbRowHeaderId  
dbFindRowheaderId(  
    dbCellViewId    cvId,  
    String           siteDefName  
);
```

Description

Searches the specified cellview for a rowHeader with the specified master siteDef name.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview.
<i>siteDefName</i>	A text string containing the name of the <code>siteDef</code> .

Return Values

<code>dbRowHeaderId</code>	Returns a valid rowHeader found in the specified cellview. Returns <code>dbcNullId</code> otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetRowHeaderSiteDef

```
dbSiteDefId  
dbGetRowHeaderSiteDef(  
    dbRowHeaderId    rowHeaderId  
);
```

Description

Gets the `siteDef` associated with the `rowHeader`.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.

Return Values

<code>dbSiteDefId</code>	Returns the <i>dbSiteDefId</i> if the <code>rowHeader</code> was successfully retrieved.

dbGetRowHeaderSiteDefName

Boolean

```
dbGetRowHeaderSiteDefName (
    dbRowHeaderId    rowHeaderId,
    String            *siteDefName
);
```

Description

Gets the `siteDefName` attribute associated with the `rowHeader` object.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.
<i>*siteDefName</i>	The name of the <code>siteDef</code> associated with the <code>rowHeader</code> . A string pointer which is assigned to the <i>siteDefName</i> returned.

Return Values

Boolean	Returns TRUE if the name of the <code>siteDef</code> of the <code>rowHeader</code> was successfully retrieved. Returns FALSE otherwise.

dbGetRowHeaderSiteDefWidth

Boolean

```
dbGetRowHeaderSiteDefWidth(  
    dbRowHeaderId      rowHeaderId,  
    uint32_t*          siteDefWidth  
);
```

Description

Gets the width of the `siteDef` attribute associated with the `rowHeader` object.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.
<i>siteDefWidth</i>	The width of the <code>siteDef</code> associated with the <code>rowHeader</code> . The <i>siteDefWidth</i> pointer is filled by the function.

Return Values

Boolean	Returns TRUE if the width of the <code>siteDef</code> of the <code>rowHeader</code> was successfully retrieved. Returns FALSE otherwise.

dbGetRowHeaderSiteDefHeight

Boolean

```
dbGetRowHeaderSiteDefHeight (
    dbRowHeaderId      rowHeaderId,
    uint32_t*          siteDefHeight
);
```

Description

Gets the height of the `siteDef` attribute associated with the `rowHeader` object.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.
<i>siteDefHeight</i>	The height of the <code>siteDef</code> associated with the <code>rowHeader</code> . The <i>siteDefHeight</i> pointer is filled by the function.

Return Values

Boolean	Returns TRUE if the height of the <code>siteDef</code> of the <code>rowHeader</code> was successfully retrieved. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetRowHeaderBBox

Boolean

```
dbGetRowHeaderBBox (
    dbRowHeaderId    rowHeaderId,
    dbBBox*           rowHeaderBBox
);
```

Description

Gets the bBox of all the rows that refer to this `rowHeader`.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.
<i>rowHeaderBBox</i>	The bBox of all the rows that refer to the row Header. The <i>rowHeaderBBox</i> pointer is filled by the function.

Return Values

Boolean	Returns TRUE if the BBox of the <code>rowHeader</code> was successfully retrieved. Returns FALSE otherwise.

dbGetRowHeaderNumRows

Boolean

```
dbGetRowHeaderNumRows (
    dbRowHeaderId    rowHeaderId,
    uint32_t*        numRows
);
```

Description

Gets the number of rows that refer to the `rowHeader` object.

Arguments

<i>rowHeaderId</i>	The database <code>dbRowHeaderId</code> for the <code>rowHeader</code> object.
<i>numRows</i>	The number of rows that refer to the <code>rowHeader</code> . The <i>numRows</i> pointer is filled by the function.

Return Values

Boolean	Returns TRUE if the number of rows that refer to the <code>rowHeader</code> was successfully retrieved. Returns FALSE otherwise.

dbIsRowHeaderBound

Boolean

```
dbIsRowHeaderBound(  
    dbRowHeaderId    rowHeaderId  
);
```

Description

Checks whether the rowHeader is bound. If the rowHeader is bound, the siteDef is read in the linkage from the rowHeader and is established by traversing from a row to that siteDef.

Arguments

<i>rowHeaderId</i>	The database dbRowHeaderId for the rowHeader object.

Return Values

Boolean	Returns TRUE if the rowHeader object is bound. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenRowHeader

```
dbGenStateId  
dbStartGenRowHeader(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the rowHeaders for a cellview.

Arguments

<i>cvId</i>	The database dbCellViewId for the cellview.

Return Values

dbGenStateId	Returns the generator dbGenStateId used to track the state of rowHeader generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenRowHeader

Boolean

```
dbGenRowHeader (
    dbGenStateId      pState,
    dbRowHeaderId*    pRowHeaderId
);
```

Description

Generates the next rowHeader in the cellview.

Arguments

<i>pState</i>	The generator dbGenStateId used to track the state of rowHeader generation.
<i>pRowHeaderId</i>	The dbRowHeaderId pointer. Filled by the function if a rowHeader is generated.

EEQ Masters

The EEQmaster (electrically equivalent master) is stored as a string on the cellview. EEQ cellviews are devices such as OR-gates or inverters that have several implementations with different shapes, geometries, and orientations.

dbGetCellViewEEQMaster

String

```
dbGetCellViewEEQMaster(  
    dbCellViewId    cvId  
);
```

Description

Returns the value of the EEQmaster attribute of the cellview. If the string is empty, NULL will be returned.

dbSetCellViewEEQMaster

Boolean

```
dbSetCellViewEEQMaster(  
    dbCellViewId    cvId,  
    String           EEQMaster  
);
```

Set the value of the EEQmaster attribute of the cellview.

Steiners

A steiner is a logical point that can represent a pin, or the start and/or end points of a route. A steiner is a figure, specifically a connFig, that can have connectivity. It can also be associated with a specific layer.

The database Id of a steiner is the following.

```
typedef dbId dbSteinerId;
```

A steiner record is the same as a figure record. See the following.

```
typedef struct dbFigRec {
    dbType          type;
    dbFigId         fig;
    size_t          size;
    dbCellViewId    cellView;
    dbNetId         net;
    dbPinId         pin;
    dbBBox          bBox;
    dbPurpose       purpose;
} dbFigRec;

typedef dbFigRec dbSteinerRec;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsSteinerId

Boolean

```
dbIsSteinerId(  
    dbId          objId  
);
```

Description

Determines whether the *dbId* is a valid steiner object.

Arguments

<i>objId</i>	The database ID of the steiner object.

Return Values

Boolean	Returns TRUE if the <i>dbId</i> is a valid steiner object. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteSteiner

Boolean

```
dbDeleteSteiner(  
    dbSteinerId      steinerId  
);
```

Description

Deletes the *dbSteinerId* object.

Arguments

<i>steinerId</i>	The database ID of the steiner object.

Return Values

<i>Boolean</i>	Returns TRUE if the <i>steinerId</i> is deleted. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateSteiner

```
dbSteinerId
dbCreateSteiner(
    dbCellViewId      cvId,
    dbBBox             *bBox,
    dbLayer            layerNum
);
```

Description

Creates a steiner object with the specified attributes.

Arguments

<i>cvId</i>	The database ID of the cellview.
<i>*bBox</i>	A pointer to the bounding box for the steiner.
<i>layerNum</i>	The layer with which to associate the steiner. If <i>layerNum</i> is <code>dbcNullLayer</code> , the steiner is not on a specific layer.

Return Values

Boolean	Returns <i>dbSteinerId</i> if the steiner is created. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetSteinerBBox

```
dbBBox *  
dbGetSteinerBBox(  
    dbSteinerId      steinerId  
);
```

Description

Gets the bounding box of the steiner object.

Arguments

<i>steinerId</i>	The database ID of the steiner.

Return Values

<i>dbBBox *</i>	Returns the bounding box of the steiner.

dbSetSteinerBBox

Boolean

```
dbSetSteinerBBox(  
    dbSteinerId      steinerId,  
    const dbBBox     *bBox  
);
```

Description

Sets the bounding box of the steiner object.

Arguments

<i>steinerId</i>	The database ID of the steiner.
<i>*bBox</i>	The bounding box of the steiner.

Return Values

Boolean	Returns TRUE if the steiner bounding box is set successfully. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetSteinerLayerHeader

```
dbLayerHeaderId  
dbGetSteinerLayerHeader(  
    dbSteinerId      steinerId  
);
```

Description

Gets the layer header of the steiner object.

Arguments

<i>steinerId</i>	The database ID of the steiner.

Return Values

<i>dbLayer</i>	Returns the layer header of the steiner object. Returns <code>dbcNullId</code> if the steiner is not on a specific layer.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetSteinerLayerNum

```
dbLayer
dbGetSteinerLayerNum(
    dbSteinerId      steinerId
);
```

Description

Gets the layer number on which the steiner object resides.

Arguments

<i>steinerId</i>	The database ID of the steiner.

Return Values

<i>dbLayer</i>	Returns the layer on which the steiner object resides. Returns <i>dbcNullLayer</i> if the steiner is not on a specific layer.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetSteinerLayerNum

Boolean

```
dbSetSteinerLayerNum(  
    dbSteinerId      steinerId,  
    dbLayer          layerNum  
);
```

Description

Sets the layer number on which the steiner resides.

Arguments

<i>steinerId</i>	The database ID of the steiner.
<i>layerNum</i>	The layer number to associate with the steiner.

Return Values

Boolean	Returns TRUE if the steiner layer is set successfully. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenSteiner

```
dbGenStateId  
dbStartGenSteiner(  
    dbCellViewId    cvId  
)
```

Description

Starts generating all the steiners in the cellview.

Arguments

<i>cvId</i>	The cellview ID for which to generate the steiner objects.

Return Values

<i>dbGenStateId</i>	The generator <code>dbGenStateId</code> which keeps track of the state of steiner generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenSteiner

Boolean

```
dbGenSteiner(  
    dbGenStateId    pState,  
    dbSteinerId     *steinerId  
)
```

Description

Generates the next steiner in the cellview.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to keep track of the state of steiner generation.
<i>*steinerId</i>	Pointer to be filled with the next steiner in the cellview.

Return Values

Boolean	Returns TRUE if the steiner object is generated. Returns FALSE otherwise.

dbStartGenSteinerToRoute

```
dbGenStateId  
dbStartGenSteinerToRoute(  
    dbSteinerId    steinerId  
)
```

Description

Starts generation of the routes associated with the specified steiner object.

Arguments

<i>steinerId</i>	The steiner ID for which to generate the routes.

Return Values

<i>dbGenStateId</i>	The generator <code>dbGenStateId</code> which keeps track of the state of route generation for the steiner object.

dbGenSteinerToRoute

Boolean

```
dbGenSteinerToRoute(  
    dbGenStateId    pState,  
    dbRouteId       *routeId  
)
```

Description

Generates the next route for the specified steiner.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> which keeps track of the state of route generation for the steiner objects.
<i>routeId</i>	Pointer to be filled with the next route for the steiner.

Return Values

Boolean	Returns TRUE if the route is generated. Returns FALSE otherwise.

dbCreateTapSteiner

```
dbSteinerId  
dbCreateTapSteiner(  
    dbCellViewId cvId,  
    const dbBBox *bbox,  
    dbLayer layerNum  
)
```

Description

Creates a Tap Steiner object with the specified bounding box in the given cellview.

Arguments

<i>cvId</i>	Database ID of a cellview.
<i>bbox</i>	List of bBox for the Tap Steiner object that will be created.
<i>layerNum</i>	Layer associated with the created Tap Steiner object that will be created.

Return Value

<i>d_steinerId</i>	ID of the created Tap Steiner.

dbIsTapSteiner

Boolean

```
dbIsTapSteinerId(  
    dbSteinerId steinerId  
)
```

Description

Checks whether the specified ID is a Tap Steiner ID.

Arguments

<i>steinerId</i>	ID of an object.

Return Value

<i>Boolean</i>	TRUE, if the object is a Tap Steiner object. FALSE, if the object is not a Tap Steiner object.

Track Patterns

This section describes functions that affect trackPattern objects.

About trackPatterns

trackPatterns are used to define the routing grid for gridded routers.

dbIsTrackPatternId

Boolean

```
dbIsTrackPatternId(  
    dbTrackPatternId    trackPatternId  
);
```

Description

Checks that the given `trackPatternId` refers to a `trackPattern` object.

Arguments

<i>trackPatternId</i>	The database <code>dbTrackPatternId</code> for the track pattern to be verified.

Return Values

Boolean	Returns TRUE if the <i>trackPatternId</i> refers to a track pattern. Returns FALSE if it is not a track pattern.

dbCreateTrackPattern

```
dbTrackPatternId
dbCreateTrackPattern(
    dbCellViewId    cvId,
    Boolean          isHorizontal,
    dbCoord          startCoord,
    unsigned int     numTracks,
    dbDistance       trackSpace,
    dbLayer          layer
);
```

Description

Creates a trackPattern with the specified attributes in the specified cell view.

Arguments

<i>cvId</i>	The database <code>dbCellViewId</code> for the cellview.
<i>isHorizontal</i>	TRUE if the tracks should be horizontal, otherwise FALSE (vertical).
<i>startCoord</i>	The starting coordinate for the trackPattern.
<i>numTracks</i>	The number of lines for the trackPattern.
<i>trackSpace</i>	The space value between two track lines.
<i>layer</i>	The <code>layer</code> to be associated with the trackPattern. If the trackPattern is not assigned to any layer, the parameter should be <code>dbcNullLayer</code> .

Return Values

<i>dbTrackPatternId</i>	Returns the database <code>dbTrackPatternId</code> for the track pattern if a track pattern is created. Returns FALSE otherwise.

dbDeleteTrackPattern

Boolean

```
dbDeleteTrackPattern(  
    dbTrackPatternId    tpId  
);
```

Description

Deletes the track pattern specified by `tpId`.

Arguments

<i>tpId</i>	The database <code>dbTrackPatternId</code> for the track pattern to be deleted.

Return Values

Boolean	Returns TRUE if the track pattern was successfully deleted. Returns FALSE otherwise.

dbGetTrackPatternIsHorizontal

Boolean

```
dbGetTrackPatternIsHorizontal(  
    dbTrackPatternId    tpId  
);
```

Description

Verifies if the track pattern specified by *tpId* is horizontal.

Arguments

<i>tpId</i>	The database <code>dbTrackPatternId</code> for the track pattern to be verified.

Return Values

Boolean	Returns TRUE if the track pattern is horizontal. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetTrackPatternStart

Boolean

```
dbGetTrackPatternStart (
    dbTrackPatternId    tpId,
    dbCoord              *start
);
```

Description

Gets the start coordinate for the track pattern specified by *tpId*.

Arguments

<i>tpId</i>	The database <code>dbTrackPatternId</code> for the trackPattern.
<i>*start</i>	A pointer to the start coordinate of the trackPattern. Returned if the start coordinate is retrieved.

Return Values

Boolean	Returns TRUE if the start coordinate of the trackPattern is successfully retrieved. Returns FALSE otherwise.

dbSetTrackPatternStart

Boolean

```
dbSetTrackPatternStart(  
    dbTrackPatternId    tpId,  
    dbCoord              start  
);
```

Description

Sets the start coordinate for the trackPattern specified by *tpId*.

Arguments

<i>tpId</i>	The database dbTrackPatternId for the trackPattern.
<i>start</i>	The start coordinate value to be set for the trackPattern specified by <i>tpId</i> .

Return Values

Boolean	Returns TRUE if the start coordinate for the track pattern specified by <i>tpId</i> was successfully set. Returns FALSE otherwise.

dbGetTrackPatternNumTracks

Boolean

```
dbGetTrackPatternNumTracks(  
    dbTrackPatternId    tpId,  
    unsigned int         *numTracks  
);
```

Description

Gets the number of track lines for the track pattern specified by *tpId*.

Arguments

<i>tpId</i>	The database <code>dbTrackPatternId</code> for the track pattern.
<i>*numTracks</i>	A pointer to the number of track lines of the trackPattern. Returned if the number of track lines is retrieved.

Return Values

Boolean	Returns TRUE if the number of track lines for the track pattern specified by <i>tpId</i> was successfully retrieved. Returns FALSE otherwise.

dbSetTrackPatternNumTracks

Boolean

```
dbSetTrackPatternNumTracks(  
    dbTrackPatternId    tpId,  
    unsigned int         num  
);
```

Description

Sets the number of track lines for the track pattern specified by *tpId*.

Arguments

<i>tpId</i>	The database dbTrackPatternId for the trackPattern.
<i>num</i>	The number of track lines to be set for the trackPattern.

Return Values

Boolean	Returns TRUE if the number of track lines for the track pattern specified by <i>tpId</i> was successfully set. Returns FALSE otherwise.

dbGetTrackPatternSpace

Boolean

```
dbGetTrackPatternSpace(  
    dbTrackPatternId    tpId,  
    dbDistance          *space  
);
```

Description

Returns the tracking space between the track lines of the track pattern specified by *tpId*.

Arguments

<i>tpId</i>	The database <code>dbTrackPatternId</code> for the trackPattern.
<i>*space</i>	A pointer to the tracking space of the trackPattern. Returned if the tracking space is retrieved.

Return Values

Boolean	Returns TRUE if the tracking space for the trackPattern specified by <i>tpId</i> was successfully retrieved. Returns FALSE otherwise.

dbSetTrackPatternSpace

Boolean

```
dbSetTrackPatternSpace(  
    dbTrackPatternId    tpId,  
    dbDistance           space  
);
```

Description

Sets the tracking space between track lines for the trackPattern specified by *tpId*.

Arguments

<i>tpId</i>	The database dbTrackPatternId for the trackPattern.
<i>space</i>	The tracking space between two track lines to be set for the trackPattern specified by <i>tpId</i> .

Return Values

Boolean	Returns TRUE if the tracking space between two track lines for the trackPattern specified for <i>tpId</i> was successfully set. Returns FALSE otherwise.

dbGetTrackPatternRoutingLayer

Boolean

```
dbGetTrackPatternRoutingLayer(  
    dbTrackPatternId    tpId,  
    dbLayer             *layer  
);
```

Description

Gets the routing layer for the trackPattern specified by *tpId*.

Arguments

<i>tpId</i>	The database dbTrackPatternId for the trackPattern.
<i>*layer</i>	A pointer to the routing layer for the trackPattern. Returned if the layer is retrieved.

Return Values

Boolean	Returns TRUE if the routing layer for the trackPattern specified for <i>tpId</i> was successfully retrieved. Returns FALSE otherwise.

dbSetTrackPatternRoutingLayer

Boolean

```
dbSetTrackPatternRoutingLayer(  
    dbTrackPatternId    cvId,  
    dbLayer              layer  
);
```

Description

Sets the routing layer for the trackPattern specified by *tpId*.

Arguments

<i>tpId</i>	The database dbTrackPatternId for the trackPattern.
<i>layer</i>	The routing layer to be set for the trackPattern specified by <i>tpId</i> .

Return Values

Boolean	Returns TRUE if the routing layer for the trackPattern specified for <i>tpId</i> was successfully set. Returns FALSE otherwise.

dbStartGenTrackPattern

```
dbGenStateId  
dbStartGenTrackPattern(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the trackPatterns for a cellview.

Arguments

<i>cvId</i>	The database dbCellViewId for the cellview.

Return Values

dbGenStateId	Returns the generator dbGenStateId used to track the state of track pattern generation.

dbGenTrackPattern

Boolean

```
dbGenTrackPattern(  
    dbGenStateId      pState,  
    dbTrackPatternId  *tpId  
);
```

Description

Generates the next track pattern in the subject cellview.

Arguments

<i>pState</i>	The generator <code>dbGenStateId</code> used to track the state of trackPattern generation.
<i>*tpId</i>	A pointer to the trackPattern ID. Returned if a trackPattern is successfully generated.

Return Values

Boolean	Returns TRUE if a trackPattern is generated. Returns FALSE otherwise.

Shape Figure Functions

Hierarchy of Shapes

Different geometrical shapes in the database, such as rectangles or polygons, are grouped into different object types. These object types collectively form the object class called shape. The following table illustrates the shape hierarchy.

Class Hierarchy for Shape

Name	Enumerated constant	ID Type for type dbType
Shape	–	dbShapeId
Rectangle	dbcRectType	dbRectId
Polygon	dbcPolygonType	dbPolygonId
Line	dbcLineType	dbLineId
Path	dbcPathType	dbPathId
Label	dbcLabelType	dbLabelId
Ellipse	dbcEllipseType	dbEllipseId
Arc	dbcArcType	dbArcId
Donut	dbcDonutType	dbDonutId
Dot	dbcDotType	dbDotId
TextDisplay	dbcTextDisplayType	dbTextDisplayId

Operations on Generic Shapes

The following operations perform operations on or retrieve information about shapes of all kinds contained in a cellview, all shapes of a particular kind contained in a cellview, or a single specified shape.

dbIsShapelId

Boolean

```
dbIsShapeId(  
    dbShapeId shapeId  
);
```

Returns TRUE if the ID passed as `shapeId` is a shape. Otherwise, it returns FALSE.

dbStartGenShape

```
dbGenStateId  
dbStartGenShape (  
    dbCellViewId cellViewId  
);
```

Starts the generator for all shapes (including both parent and child shapes) contained in a cellview. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions.”](#)

dbGenShape

```
Boolean  
dbGenShape (  
    dbGenStateId stateId, dbShapeId pShapeId  
);
```

Generates all shapes (including both parent and child shapes) contained in a cellview. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions.”](#)

dbStartGenNonChildShape

```
dbGenStateId  
dbStartGenNonChildShape (  
    dbCellViewId cellViewId  
);
```

Starts the generator for all shapes (including both parent and child shapes) contained in a cellview. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions.”](#)

dbGenNonChildShape

```
Boolean  
dbGenNonChildShape (  
    dbGenStateId stateId, dbShapeId pShapeId  
);
```

Generates all shapes contained in the cellview, specified by `cellViewId`, that are not child shapes of some parent. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbStartGenLPToShape

```
dbGenStateId  
dbStartGenLPToShape(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose  
);
```

Starts the generator for all shapes (including both parent and child shapes) contained in a cellview with the specified layer and purpose. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbGenLPToShape

```
Boolean  
dbGenLPToShape(  
    dbGenStateId stateId, dbShapeId pShapeId  
);
```

Generates all shapes (including both parent and child shapes) contained in a cellview with the specified layer and purpose. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbStartGenLPToPinShape

```
dbGenStateId  
dbStartGenLPToPinShape(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose  
);
```

Starts the generator for all pin shapes (including both parent and child shapes) contained in a cellview with the specified layer and purpose. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbGenLPToPinShape

```
Boolean  
dbGenLPToPinShape(  
    dbGenStateId stateId, dbShapeId pShapeId  
);
```

Generates all pin shapes (including both parent and child shapes) contained in a cellview with the specified layer and purpose. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbStartGenLPToNonChildShape

dbGenStateId

```
dbStartGenLPToNonChildShape(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose  
);
```

Starts the generator for all the shapes contained in the cellview with the specified layer and purpose that are not child shapes of some parent. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbGenLPToNonChildShape

Boolean

```
dbGenLPToNonChildShape(  
    dbGenStateId stateId, dbShapeId pShapeId  
);
```

Generates all the shapes contained in the cellview with the specified layer and purpose that are not child shapes of some parent. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbDeleteShape

Boolean

```
dbDeleteShape(  
    dbShapeId shapeId  
);
```

Deletes the shape specified by `shapeId`. After this function is called, `shapeId` is invalid and you cannot use it. If the shape specified by `shapeId` is a parent, then all descendant figures are deleted as well. If the shape specified by `shapeId` is a child and the child's point list matches the parent's point list, the shape cannot be deleted, and the function returns FALSE. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbMoveShape

dbShapeId

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbMoveShape (  
    dbShapeId shapeId, dbCellViewId destCellViewId, dbTransform transform  
);
```

Moves the shape specified by `shapeId`. The shape is magnified, then rotated and reflected about the origin as specified by the transformation's orientation. Then it is translated by the transformation's offset. If the shape, cellview, or transformation is invalid, the function gives a fatal error message and returns `dbcNullShapeId`. Otherwise, the function returns the ID of the transformed shape.

If `destCellViewId` is `dbcNullCellViewId` or if the `destCellViewId` is the cellview that contains the shape, the move is done within the same cellview. The shape's ID and purpose are not changed. All properties on the shape are preserved.

If the `destCellViewId` is not the original shape's cellview, the shape is deleted and a new shape is created in the cellview specified by `destCellViewId`. The newly created shape has a different ID but retains its original purpose. The new shape is not associated with any net, pin, or terminal of the original shape. All properties on the shape are preserved but have new IDs.

If `shapeId` refers to a parent shape with children, all descendant figures are moved according to the information kept in `transform`. If `shapeId` refers to a child by itself, the shape is moved without moving the parent. However, if the move is between cellviews, the child is moved but is no longer a child in the destination cellview. All other features regarding purpose and properties apply to child as well as parent shapes that are moved.

If the shape specified by `shapeId` is a child and the child's point list matches the parent's point list, the move is not allowed and the function returns `dbcNullShapeId`. For more information on parent-child relationships, see [Chapter 9, "CDBA Parent-Child Relationship Functions."](#)

dbCopyShape

```
dbShapeId  
dbCopyShape (  
    dbShapeId shapeId, dbCellViewId destCellViewId, dbTransform transform  
);
```

Copies the shape specified by `shapeId`. The copied shape is magnified, then rotated and reflected about the origin as specified by the transformation's orientation. Then it is translated by the transformation's offset. If the shape, cellview, or transformation is invalid, the function gives a fatal error message and returns `dbcNullShapeId`. Otherwise, it returns the ID of the copied shape.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

If `destCellViewId` is `dbcNullCellViewId`, the shape is copied in the same cellview. If `destCellViewId` is not `dbcNullCellViewId`, the shape is copied into the cellview specified by `destCellViewId`.

The new shape retains its purpose and properties, and the figure is not associated with any net, pin, or terminal of the original shape.

If `shapeId` refers to a parent shape with children, all descendant figures are copied according to the information kept in `transform`. If `shapeId` refers to a child by itself, the shape is copied without copying the parent. However, if the copy is between cellviews, the child is copied but no longer is a child in the destination cellview. All other features regarding purpose and properties apply to child as well as parent shapes that are copied.

If the shape specified by `shapeId` is a child and the child's point list matches the parent's point list, the copy is not allowed and the function returns `dbcNullShapeId`. For more information on parent-child relationships, see [Chapter 9, "CDBA Parent-Child Relationship Functions."](#)

dbGetShapeLP

Boolean

```
dbGetShapeLP(  
    dbShapeId shapeId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose on which the shape specified by `shapeId` was created.

dbGetShapeBBox

dbBBox *

```
dbGetShapeBBox(  
    dbShapeId shapeId  
);
```

Returns a pointer to the bounding box of the shape specified by `shapeId`. This `bBox` structure is read only and is overwritten by the next call to `dbGet*BBox` or if the shape is deleted or modified. If you want to keep a permanent copy of the data, you must allocate your own storage and copy the data into it.

dbSetShapeLayer

Boolean

```
dbSetShapeLayer(  
    dbShapeId shapeId, dbLayer layer  
);
```

Sets the layer of the shape.

dbSetShapePurpose

Boolean

```
dbSetShapePurpose(  
    dbShapeId shapeId, dbPurpose purpose  
);
```

Sets the purpose of the shape.

dbSetViaShapePurpose

```
dbSetViaShapePurpose(  
    dbViaId via, dbPurpose purpose  
);
```

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 this function with `dbcUnknownPurpose` as the new purpose value.

Note: The specified `via` must be a custom `cdsVia`.

dbSetShapeLP

Boolean

```
dbSetShapeLP(  
    dbShapeId shapeId, dbLayer layer, dbPurpose purpose  
);
```

Sets the layer and purpose of the shape.

dbHasShape

Boolean

```
dbHasShape(  
    dbCellViewId cellViewId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Checks if the `cellViewId` has any shapes. If it has any shapes, this function returns TRUE. Otherwise, the function returns FALSE.

dbLPHasShape

Boolean

```
dbLPHasShape(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose  
);
```

Returns TRUE if this `layer` and `purpose` pair has any shapes. The function returns FALSE if this layer-purpose pair has no shape.

dbGetShapeCellView

dbCellViewId

```
dbGetShapeCellView(  
    dbShapeId shapeId  
);
```

Gets the `cellViewId` of the shape.

Rectangles

A rectangle in the OpenAccess database is assumed to be aligned with the x and y axes. For storage efficiency, each rectangle is represented by its bounding box.

dbCreateRect

```
dbRectId  
dbCreateRect(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbBBox bBox  
);
```

Creates a rectangle in the specified cellview on the specified layer-purpose with the lower left and upper right corners defined by the `bBox`. It returns the ID of the newly created rectangle. You cannot create shapes on a layer-purpose whose purpose is `dbcAllPurpose`.

dbSetRectBBox

```
Boolean  
dbSetRectBBox(  
    dbRectId rectId, dbBBox bBox  
);
```

Sets the bounding box of the rectangle. If the rectangle is a parent with matching points, then its descendants' points are set as well. If the rectangle is a child and the child's point list matches the parent's point list, you cannot set the bounding box of the rectangle.

dbDeleteRect

```
Boolean  
dbDeleteRect(  
    dbRectId rectId  
);
```

Deletes the rectangle specified by `rectId`. For other effects of deleting shapes, see [“dbDeleteShape”](#) on page 415.

dbGetRectLP

```
Boolean  
dbGetRectLP(  
    dbRectId rectId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Retrieves the layer and purpose of the rectangle by using the second and third arguments, respectively.

dbGetRectBBox

```
dbBBox      *
dbGetRectBBox (
    dbRectId rectId
);
```

Returns a pointer to the bounding box of the rectangle. This `bBox` structure is read only and is overwritten by the next call to `dbGet*BBox`.

dbConvertRectToPolygon

```
dbPolygonId
dbConvertRectToPolygon (
    dbRectId rectId
);
```

Converts the rectangle to a polygon. The `rectId` is kept with the converted polygon ID. If the conversion fails, the function returns `dbcNullPolygonId`. If the rectangle has match points set, the conversion fails. The conversion cannot be done.

Polygons

A polygon in the OpenAccess database is represented by an array of points specifying its vertices. Although self-intersecting polygons can be represented in the database, many of the Cadence tools do not handle them.

dbCreatePolygon

dbPolygonId

```
dbCreatePolygon(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, unsigned int  
    nPoints, dbPoint pointArray  
);
```

Creates a polygon in the specified cellview with the specified layer-purpose. `nPoints` is the number of vertices in the polygon, and `pointArray` is an array of points containing that number of vertices. You cannot create shapes on a layer whose purpose is `dbcAllPurpose`.

dbSetPolygonPoint

Boolean

```
dbSetPolygonPoint(  
    dbPolygonId polygonId, unsigned int nPoints, dbPoint pointArray  
);
```

Sets the number of points and point arrays of the polygon. If the polygon is a parent with matching points, its descendants' points are set as well. If the polygon is a child and the child's point list matches the parent's point list, you cannot set the points. For more information on parent-child relationships, see [Chapter 9, "CDBA Parent-Child Relationship Functions."](#)

dbDeletePolygon

Boolean

```
dbDeletePolygon(  
    dbPolygonId polygonId  
);
```

Deletes the polygon specified by `polygonId`. For other effects of deleting a shape, see `dbDeleteShapevertexes`.

dbGetPolygonLP

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetPolygonLP(  
    dbPolygonId polygonId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose on which the polygon was created.

dbGetPolygon

Boolean

```
dbGetPolygon(  
    dbPolygonId polygonId, unsigned int pNPoints, dbPoint pPointArray  
);
```

Retrieves all vertices in the polygon. `pNPoints` returns the number of vertices, and `pPointArray` returns a pointer to an array of points representing all vertices in the polygon. This point array is read only and is overwritten by the next call that returns a point array. If you want to keep a permanent record of the vertices, you must allocate your own storage and make a copy of the point array before you call any other database routine.

dbGetPolygonBBox

dbBBox *

```
dbGetPolygonBBox(  
    dbPolygonId polygonId  
);
```

Retrieves a pointer to the bounding box of the polygon. This bounding box structure is read only and is overwritten by the next call to `dbGet *BBox`.

dbStartGenPolygonToPoint

dbGenStateId

```
dbStartGenPolygonToPoint(  
    dbPolygonId polygonId  
);
```

Starts the point generator for the polygon specified by `polygonId`. The function creates the appropriate generator state and returns a pointer to that generator state.

dbGenPolygonToPoint

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGenPolygonToPoint(  
    dbGenStateId stateId, dbPoint pPoint  
);
```

Generates each point in the polygon, step by step. `stateId` must have been created by an earlier call to `dbStartGenPolygonPoint`.

dbConvertPolygonToRect

```
dbRectId  
    dbConvertPolygonToRect(  
        dbPolygonId polygonId  
    );
```

Converts a polygon with four points to a rectangle. If the polygon cannot be converted to a rectangle, `dbcNullRectId` is returned. If the polygon has match points set, it cannot be converted.

Lines

A line in the OpenAccess database is represented by an array of points specifying its vertices.

dbCreateLine

```
dbLineId  
dbCreateLine(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, unsigned int  
    nPoints, dbPoint pointArray  
);
```

Creates a line in the specified cellview with the specified layer and purpose. `nPoints` is the number of vertices in the line, and `pointArray` is an array of points containing that number of vertices. You cannot create shapes on a layer whose purpose is `dbcAllPurpose`.

dbSetLinePoint

```
Boolean  
dbSetLinePoint(  
    dbLineId lineId, unsigned int nPoints, dbPoint pointArray  
);
```

Sets the number of points and the point array of the line. If the line is a parent with matching points, its descendants' points are set as well. If the line is a child and the child's point list is set to match the parent's point list, you cannot set the line points. For more information on parent-child relationships, see [Chapter 9, "CDBA Parent-Child Relationship Functions,"](#).

dbDeleteLine

```
Boolean  
dbDeleteLine(  
    dbLineId lineId  
);
```

Deletes the line specified by `lineId`. For other effects of deleting a shape, see ["dbDeleteShape"](#) on page 415.

dbGetLineLP

```
Boolean
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetLineLP(  
    dbLineId lineId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose of the specified line.

dbGetLine

Boolean

```
dbGetLine(  
    dbLineId lineId, unsigned int pNPoints, dbPoint pPointArray  
);
```

Retrieves the vertices in the line. `pNPoints` returns the number of vertices, and `pPointArray` returns a pointer to an array of points representing all the vertices in the line. This point array is read only and is overwritten by the next call that returns a point array. If you want to keep a permanent record of the vertices, you must allocate your own storage and make a copy of the point array before you call any other database routine.

dbGetLineBBox

dbBBox *

```
dbGetLineBBox(  
    dbLineId lineId  
);
```

Retrieves a pointer to the bounding box of the line. This bounding box structure is read only and is overwritten by the next call to `dbGet*BBox`.

dbStartGenLineToPoint

dbGenStateId

```
dbStartGenLineToPoint(  
    dbLineId lineId)
```

Starts the point generator for the line specified by `lineId`. The function creates the appropriate generator state and returns a pointer to that generator state.

dbGenLineToPoint

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGenLineToPoint(  
    dbGenStateId stateId, dbPoint pPoint  
);
```

Generates each point in the line, step by step. You must create `stateId` by an earlier call to `dbStartGenLineToPoint`.

Paths

A path in the OpenAccess database is represented by an array of points specifying its vertices. However, unlike lines, paths have widths and can be displayed in a number of different styles. The following table lists all display styles currently supported.

The enumerated constants are grouped in the enumerated type `dbPathStyle`. The EDIF terminology (of specifying path styles in terms of their end and corner types, each of which can be `extend`, `truncate`, or `round`) is being used. Among the 16 possible combinations, four of the more useful ones are supported. Because these four styles are also known by other names, the original name is listed in parentheses.

Path Display Styles

End type	Enumerated constants corner type in type <code>dbPathStyle</code>
Truncate, Extend	<code>dbcTruncateExtend</code> (<code>SquareFlush</code>)
Extend, Extend	<code>dbcExtendExtend</code> (<code>SquareOffset</code>)
<code>varExtend</code> , <code>varExtend</code>	<code>dbcVarExtendExtend</code> (<code>varSquareOffset</code>)
Round, Round	<code>dbcRoundRound</code> (<code>OctagonEnded</code>)

`dbCreatePath`

`dbPathId`

```
dbCreatePath(
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, unsigned int
    nPoints, dbPoint pointArray, dbDistance width, dbPathStyle style
);
```

Creates a path in the specified cellview with the specified layer-purpose pair. `nPoints` is the number of vertices in the path. `pointArray` is an array of points containing that number of vertices. If the number of vertices specified in `nPoints` is two, and the `pointArray` contains the same vertices, the function returns `NULL`. `width` is the width of the path and must be an even number. Otherwise, CDBA rounds it up to the next even number. You cannot create shapes on a layer whose purpose is `dbcAllPurpose`.

`dbSetPathPoint`

`Boolean`

```
dbSetPathPoint(
    dbPathId pathId, unsigned int nPoints, dbPoint pointArray
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Sets the number of points and the point arrays of the path. If the number of vertices specified in `nPoints` is two, and the `pointArray` contains the same vertices, the function returns `FALSE`. If the path is a parent with matching points, its descendants have their points set as well. If the path is a child and the child's point list matches the parent's point list, the set is not allowed. For more information on parent-child relationships, see [Chapter 9, "CDBA Parent-Child Relationship Functions."](#)

dbSetPathExt

Boolean

```
dbSetPathExt(  
    dbPathId pathId, int pathBeginExt, int pathEndExt  
);
```

Sets the extensions of the path whose path style is `dbcVarExtendExtend`. If the path style is not `dbcVarExtendExtend`, the function fails and returns `FALSE`.

dbSetPathStyle

Boolean

```
dbSetPathStyle(  
    dbPathId pathId, dbPathStyle pathStyle  
);
```

Sets the style of the path. If the function fails, it returns `FALSE`.

dbSetPathWidth

Boolean

```
dbSetPathWidth(  
    dbPathId pathId, dbDistance width  
);
```

Sets the width of the path.

dbDeletePath

Boolean

```
dbDeletePath(  
    dbPathId pathId  
);
```

Deletes the path specified by `pathId`. For other effects of deleting a path shape, see [“dbDeleteShape”](#) on page 415.

dbGetPathLP

Boolean

```
dbGetPathLP(  
    dbPathId pathId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose of the specified path.

dbGetPath

Boolean

```
dbGetPath(  
    dbPathId pathId, unsigned int pNPoints, dbPoint pPointArray, dbDistance  
    pWidth, dbPathStyle pStyle  
);
```

Retrieves the vertices in the path. `pNPoints` returns the number of vertices, and `pPointArray` returns a pointer to an array of points representing all the vertices in the path. This point array is read only and is overwritten if `dbGetPath` is called again or if the path is deleted or modified. If you want to keep a permanent record of the vertices, you must allocate your own storage and make a copy of the point array before you call any other database routine. The width and style of the path are returned using `pWidth` and `pStyle`, respectively.

dbGetPathBBox

dbBBox *

```
dbGetPathBBox(  
    dbPathId pathId  
);
```

Retrieves a pointer to the bounding box of the path. This `bBox` structure is read only and is overwritten by the next call to `dbGet*BBox`.

dbGetPathExt

Boolean

```
dbGetPathExt(  
    dbPathId pathId, int pPathBeginExt, int pPathEndExt  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Returns the extensions of the path. If the path style is not `dbcVarExtendExtend`, then both `PathBeginExt` and `PathEndExt` are returned as 0.

dbGetPathWidth

```
dbDistance
dbGetPathWidth(
    dbPathId pathId
);
```

Returns the width of the path.

dbGetPathStyle

```
dbPathStyle
dbGetPathStyle(
    dbPathId pathId
);
```

Returns the style of the path.

dbStartGenPathToPoint

```
dbGenStateId
dbStartGenPathToPoint(
    dbPathId pathId
);
```

Starts the point generator for the path specified by `pathId`. It creates the appropriate generator state and returns a pointer to it.

dbGenPathToPoint

```
Boolean
dbGenPathToPoint(
    dbGenStateId stateId, dbPoint pPoint
);
```

Generates each point in the path, step by step. `stateId` must have been created by an earlier call to `dbStartGenPathToPoint`.

dbConvertPathToPolygon

```
dbPolygonId  
    dbConvertPathToPolygon(  
        dbPathId pathId  
    );
```

Converts the path to a polygon. The `pathId` is kept with the converted polygon ID. If the conversion fails, the function returns `dbcNullPolygonId`. If the path has match points set, you cannot convert the path.

dbConvertLineToPath

```
dbPathId  
    dbConvertLineToPath(  
        dbLineId lineId, dbDistance width, dbPathStyle style  
    );
```

Converts a line to a path. The `lineId` is kept with the converted path ID. If the conversion fails, the function returns `dbcNullPathId`.

Labels

Labels are character strings with the following attributes:

- Drafting
- Font height
- Font style
- Justify
- Label type
- Origin
- Overbar
- Rotation

Drafting

Specifies as Boolean whether or not the label stays backward or upside-down. If the drafting argument is set to TRUE, the backward or upside-down label is displayed in a readable form. If the drafting argument is set to FALSE, the backward or upside-down label stays the way it is.

dbCreateLabel

dbLabelId

```
dbCreateLabel(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbPoint origin,  
    String labelString, dbLabelJustify justification, dbOrient rotation,  
    dbFontStyle fontStyle, dbDistance fontHeight  
);
```

Creates the label with the specified attributes. When the label is created, three attributes—`labelType`, `overbar`, and `drafting`—each default to `dbcNormalLabel`, `FALSE`, and `TRUE`, respectively. If you want to change these defaults, use the `set` functions. You cannot create shapes on a layer whose purpose is `dbcAllPurpose`.

dbSetLabelType

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbSetLabelType(  
    dbLabelId labelId, dbLabelType labelType  
);
```

Sets the `labelType` of the label.

dbSetLabelFontStyle

Boolean

```
dbSetLabelFontStyle(  
    dbLabelId labelId, dbFontStyle fontStyle  
);
```

Sets the `fontStyle` of the label.

dbSetLabelOverbar

Boolean

```
dbSetLabelOverbar(  
    dbLabelId labelId, Boolean overbar  
);
```

Sets the overbar of the label.

dbSetLabelDrafting

Boolean

```
dbSetLabelDrafting(  
    dbLabelId labelId, Boolean drafting  
);
```

Sets the drafting of the label.

dbSetLabelOrigin

Boolean

```
dbSetLabelOrigin(  
    dbLabelId labelId, dbPoint origin  
);
```

Sets the origin of the label.

dbSetLabelTheLabel

Boolean

```
dbSetLabelTheLabel(  
    dbLabelId labelId, String labelString  
);
```

Sets the `labelString` of the label.

dbSetLabelJustify

Boolean

```
dbSetLabelJustify(  
    dbLabelId labelId, dbLabelJustify justification  
);
```

Sets the `justification` of the label.

dbSetLabelOrient

Boolean

```
dbSetLabelOrient(  
    dbLabelId labelId, dbOrient rotation  
);
```

Sets the `rotation` of the label.

dbSetLabelHeight

Boolean

```
dbSetLabelHeight(  
    dbLabelId labelId, dbDistance fontHeight  
);
```

Sets the `fontHeight` of the label.

dbDeleteLabel

Boolean

```
dbDeleteLabel(  
    dbLabelId labelId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Deletes the specified label. For other effects of deleting a label (shape), see “[dbDeleteShape](#)” on page 415.

dbGetLabel

Boolean

```
dbGetLabel(  
    dbLabelId labelId, String pLabelString, dbLabelJustify pJustification,  
    dbOrient pRotation, dbFontStyle pFontStyle, dbDistance pFontHeight  
);
```

Retrieves the attributes from the specified label.

dbGetLabelOrigin

dbPoint

```
dbGetLabelOrigin(  
    dbLabelId labelId  
);
```

Returns the x and y coordinates of the origin of the label in a `dbPoint` structure.

dbGetLabelLP

Boolean

```
dbGetLabelLP(  
    dbLabelId labelId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and the purpose of the label.

dbGetLabelBBox

dbBBox

```
dbGetLabelBBox(  
    dbLabelId labelId  
);
```

Returns a pointer to the bounding box of the label. This `bBox` structure is read only and is overwritten by the next call to `dbGet*BBox`.

dbGetLabelType

dbLabelType

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetLabelType(  
    dbLabelId labelId  
);
```

Returns the label type.

dbGetLabelOverbar

Boolean

```
dbGetLabelOverbar(  
    dbLabelId labelId  
);
```

Returns TRUE if the label has `overbar` turned on. The function returns FALSE if the label has `overbar` turned off.

dbGetLabelDrafting

Boolean

```
dbGetLabelDrafting(  
    dbLabelId labelId  
);
```

Returns TRUE if the label has `drafting` turned on. The function returns FALSE if the label has `drafting` turned off.

Font Height

The font height is the height of the font to be used as an unsigned integer.

Font Style

Font style is the style of font to use for the label. Some of the more exotic font styles listed in the following table might not be available on all systems. The different styles are grouped in the enumerated type `dbFontStyle`.

Font Styles

Name	Enumerated constants in type <code>dbFontStyle</code>
European style	<code>dbcEuroStyleFont</code>
Gothic	<code>dbcGothicFont</code>

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Font Styles

Name	Enumerated constants in type dbFontStyle
Mathematics	dbcMathFont
Roman	dbcRomanFont
Script	dbcScriptFont
Stick	dbcStickFont
Fixed	dbcFixedFont
Swedish	dbcSwedishFont

Justify

Justify refers to the origin of the label relative to the text before orientation. The following table shows the nine possible justifications, which are grouped in the enumerated type `dbLabelJustify`.

Label Justifications

Enumerated constants for type dbLabelJustify			
	Left	Center	Right
Upper	dbcUpperLeft	dbcUpperCenter	dbcUpperRight
Center	dbcCenterLeft	dbcCenterCenter	dbcCenterRight
Lower	dbcLowerLeft	dbcLowerCenter	dbcLowerRight

Label Type

Label type controls whether or not and how a label must be evaluated. The following table shows the three possible types.

Label Types

Label type	Enumerated constants in type dbLabelType
Normal label	dbcNormalLabel
NLP label	dbcNLPLabel

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Label Types

Label type	Enumerated constants in type <code>dbLabelType</code>
IL label	<code>dbcILLLabel</code>

Origin

The origin is the point where the label is placed.

Overbar

Specifies as Boolean whether the label has the under bar character (`_`) that needs be converted to an overbar. If the overbar argument is set to `TRUE`, the function converts the under bar character (`_`) to an over bar when it is encountered. If the over bar argument is set to `FALSE`, then the under bar character (`_`) remains.

Rotation

Rotation refers to how the label must be rotated when it is displayed. The following table shows the eight possible rotations.

Label Rotations

Label orientation	Enumerated constants in type <code>dbOrient</code>
No rotation	<code>dbcR0</code>
Rotate 90 degrees	<code>dbcR90</code>
Rotate 180 degrees	<code>dbcR180</code>
Rotate 270 degrees	<code>dbcR270</code>
Mirror about the y axis	<code>dbcMY</code>
Mirror about the y axis, then rotate 90 degrees	<code>dbcMYR90</code>
Mirror about the x axis	<code>dbcMX</code>
Mirror about the x axis, then rotate 90 degrees	<code>dbcMXR90</code>

Ellipses

An ellipse in the OpenAccess database is defined by its bounding box. A circle is a special case of an ellipse where the bounding box is a square. Because rectangles are also defined by their bounding boxes, the functions for manipulating ellipses look like the functions for manipulating rectangles.

dbCreateEllipse

```
dbEllipseId  
dbCreateEllipse(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbBBox bBox  
);
```

Creates an ellipse in the specified cellview on the specified layer-purpose pair. The lower left and the upper right corners define the bounding box specified by `bBox`. It returns the ID of the newly created ellipse. You cannot create shapes on a layer whose purpose is `dbcAllPurpose`.

dbSetEllipseBBox

```
Boolean  
dbSetEllipseBBox(  
    dbEllipseId ellipseId, dbBBox bBox  
);
```

Sets the bounding box of the ellipse.

dbDeleteEllipse

```
Boolean  
dbDeleteEllipse(  
    dbEllipseId ellipseId  
);
```

Deletes the ellipse specified by `ellipseId`. For other effects of deleting shapes, see [dbDeleteShape](#) on page 415.

dbGetEllipseLP

```
Boolean
```


Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetEllipseLP(  
    dbEllipseId ellipseId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose of the ellipse by using the second and third arguments, respectively.

dbGetEllipseBBox

```
dbBBox *  
dbGetEllipseBBox(  
    dbEllipseId ellipseId  
);
```

Returns a pointer to the bounding box of the ellipse. This bounding box structure is read only and is overwritten by the next call to `dbGet*BBox`.

dbConvertEllipseToPolygon

```
dbPolygonId  
dbConvertEllipseToPolygon(  
    dbEllipseId ellipseId int *pNSides  
);
```

Converts the ellipse to a polygon with the number of sides specified by `pNSides`. If the conversion fails, the function returns `dbcNullPolygonId`. Otherwise, the function returns the converted polygon ID. You must initialize `pNSides` to specify the number of points of the sides expected. However, this function might convert the ellipse to the polygon with a different number of points, so the value is passed back through `pNSides`.

Arcs

Conceptually, an arc in the OpenAccess database can be considered an ellipse augmented by two angles: the angle at which the arc begins and the angle at which the arc ends. Arcs with a sweep angle larger than π radians (180 degrees) are not currently supported. This means only arcs up to the size of half an ellipse can be represented in the database.

dbCreateArc

dbArcId

```
dbCreateArc(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbBBoxellipseBBox,  
    dbAngle startAngle, dbAngle stopAngle  
);
```

Creates an arc specified by the ellipse `ellipseBBox`, the starting angle `startAngle`, and the stopping angle `stopAngle`. Both angles must be specified in radians. The sweep of the arc begins at `startAngle` and proceeds counterclockwise ending at `stopAngle`. The sweep angle must be no more than π radians (180 degrees). You cannot create shapes on a layer-purpose pair whose purpose is `dbcAllPurpose`.

dbSetArc

Boolean

```
dbSetArc(  
    dbArcId arcId, dbBBox ellipseBBox, dbAngle startAngle, dbAngle stopAngle  
);
```

Sets the `ellipseBBox` and angles of the arc.

dbCreateArcByThreePoints

dbArcId

```
dbCreateArcByThreePoints(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbPoint point1,  
    dbPoint point2, dbPoint point3  
);
```

Provides an alternative way to create an arc by specifying three points through which the arc must pass. The three points must uniquely specify a well-defined arc. Otherwise, `dbCreateArcByThreePoints` returns an error.

dbSetArcPoint

Boolean

```
dbSetArcPoint(  
    dbArcId arcId, dbPoint point1, dbPoint point2, dbPoint point3  
);
```

Sets the three points of the arc.

dbDeleteArc

Boolean

```
dbDeleteArc(  
    dbArcId arcId  
);
```

Deletes the arc specified by `arcId`. For other effects of deleting an arc (shape), see [“dbDeleteShape”](#) on page 415.

dbGetArcLP

Boolean

```
dbGetArcLP(  
    dbArcId arcId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose of the arc, using the second and third arguments, respectively.

dbGetArc

Boolean

```
dbGetArc(  
    dbArcId arcId, dbBBox ppEllipseBBox, dbAngle pStartAngle, dbAngle pStopAngle  
);
```

Retrieves the three components of the arc specified by `arcId`: the bounding box of the ellipse, the starting angle, and the stopping angle. The function returns them using the second, third, and fourth arguments, respectively. Notice that the function returns a pointer to the bounding box. The bounding box structure specified by this pointer is read only and is overwritten by the next call to `dbGetArc`.

Because arcs are stored in integer database units in the database, rounding errors do occur when you specify the three components. Arcs are mapped from or to the internal data

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

structures. Therefore, `dbGetArc` might not be same as the angles you specified in the `dbCreateArc` call when you created the arc.

dbGetArcBBox

```
dbBBox *  
dbGetArcBBox(  
    dbArcId arcId  
);
```

Retrieves the real bounding box of the arc, the smallest rectangle that completely encloses the arc. The function returns a pointer to the bounding box. This bounding box is read only and is overwritten by the next call to `dbGet*BBox`.

dbGetArcBothBBox

```
Boolean  
dbGetArcBothBBox(  
    dbArcId arcId, dbBBox ppEllipseBBox, dbBBox ppRealBBox  
);
```

Retrieves the arc as two bounding boxes. Because the sweep angle of an arc must be less than `pi` radians (180 degrees), the arc can be represented by two bounding boxes: the bounding box defining the ellipse and the bounding box defining the arc within that ellipse. Both bounding boxes are read only and are overwritten by the next call to `dbGetArcBothBBox`.

dbConvertArcToLine

```
dbLineId  
dbConvertArcToLine(  
    dbArcId arcId, unsigned int pNSides  
);
```

Converts the arc to a line with the number of segments specified by `pNSides`. If the conversion fails, the function returns `dbcNullLineId`. Otherwise, the function returns the converted line ID. You initialize `pNSides` to specify the number of points expected. However, this function might convert the arc to a line with a different number of points, so the value is passed back through `pNSides`.

Donuts

A Donut in the OpenAccess database is defined by two concentric circles.

dbCreateDonut

```
dbDonutId  
dbCreateDonut(  
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbPoint center,  
    dbDistance outerRadius, dbDistance holeRadius  
);
```

Creates a donut in the specified cellview on the specified layer-purpose. The outer circle, with radius `outerRadius`, and the inner circle (the circle demarcating the hole), with radius `holeRadius`, are both centered at `center`. It returns the ID of the newly created donut. If the creation fails, `dbcNullDonutId` is returned. You cannot create shapes on a layer whose purpose is `dbcAllPurpose`.

dbSetDonutCenter

```
Boolean  
dbSetDonutCenter(  
    dbDonutId donutId, dbPoint center  
);
```

Sets the center of the donut.

dbSetDonutRadius

```
Boolean  
dbSetDonutRadius(  
    dbDonutId donutId, dbDistance outerRadius, dbDistance holeRadius  
);
```

Sets the radii of the donut.

dbDeleteDonut

```
Boolean  
dbDeleteDonut(  
    dbDonutId donutId  
);
```

Deletes the donut specified by `donutId`. If the deletion fails, the function returns `FALSE`. For other effects of deleting a donut shape, see [“dbDeleteShape”](#) on page 415.

dbGetDonutLP

Boolean

```
dbGetDonutLP(  
    dbDonutId donutId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose of the donut. If the retrieval fails, the function returns `FALSE`.

dbGetDonutBBox

dbBBox *

```
dbGetDonutBBox(  
    dbDonutId donutId  
);
```

Retrieves the bounding box of the donut. The bounding box is read only and is overwritten by the next call to `dbGet*BBox`.

dbGetDonutBothBBox

Boolean

```
dbGetDonutBothBBox(  
    dbDonutId donutId, dbBBox ppOuterBBox, dbBBox ppHoleBBox  
);
```

Retrieves the bounding boxes of the donut. These bounding box structures are read only and are overwritten by the next call to `dbGetDonutBothBBox`.

dbConvertDonutToPolygon

dbPolygonId

```
dbConvertDonutToPolygon(  
    dbDonutId donutId, unsigned int pNSides  
);
```

Converts the donut to a polygon with the number of sides specified by `pNSides`. If the conversion fails, the function returns `dbcNullPolygonId`. Otherwise, the function returns the converted polygon ID. You initialize `pNSides` to specify the number of points expected.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

However, this function might convert the donut to a polygon with a different number of points. The value is passed back through `pNSides`.

Dots

A Dot in the OpenAccess database is defined by a single point.

dbCreateDot

```
dbDotId
dbCreateDot(
    dbCellViewId cellViewId, dbLayer layer, dbPurpose purpose, dbPoint xy
);
```

Creates a dot in the specified cellview on the specified layer-purpose. The function returns the ID of the newly created dot. If the creation fails, the function returns `dbcNullDotId`. You cannot create shapes on a layer-purpose pair whose purpose is `dbcAllPurpose`.

dbSetDot

```
Boolean
dbSetDot(
    dbDotId dotId, dbPoint xy
);
```

Sets the point of the dot.

dbSetDotSize

```
Boolean
dbSetDotSize(
    dbDotId dotId, dbDistance width, dbDistance height
);
```

Sets the x direction (width) and y direction (height) of the dot.

dbDeleteDot

```
Boolean
dbDeleteDot(
    dbDotId dotId
);
```

Deletes the dot specified by `dotId`. If the deletion fails, the function returns `FALSE`. For other effects of deleting shapes, see [“dbDeleteShape”](#) on page 415.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetDotLP

Boolean

```
dbGetDotLP(  
    dbDotId dotId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the `layer` and `purpose` of the dot. If the retrieval fails, the function returns `FALSE`.

dbGetDot

dbPoint

```
dbGetDot(  
    dbDotId dotId  
);
```

Retrieves the dot.

Shape Route Status

The following functions get and set the route status of the shape.

dbGetShapeRouteStatus

Boolean

```
dbGetShapeRouteStatus (
    dbShapeId      shapeId,
    dbRouteStatus* routeStatus
);
```

Description

Returns the routeStatus of the specified shape via the routeStatus parameter.

Arguments

<i>shapeId</i>	The database ID of the shape object.
<i>routeStatus</i>	The routeStatus can be normalRouteStatus, fixedRouteStatus, or lockedRouteStatus.

Return Values

Boolean	Returns TRUE if the shapeRouteStatus was successfully retrieved. Returns FALSE otherwise.

dbSetShapeRouteStatus

Boolean

```
dbSetShapeRouteStatus (
    dbShapeId      shapeId,
    dbRouteStatus  routeStatus
);
```

Description

Sets the routeStatus for the specified shape.

Arguments

<i>shapeId</i>	The database ID of the shape object.
<i>routeStatus</i>	The current route status can be <code>normalRouteStatus</code> , <code>fixedRouteStatus</code> , or <code>lockedRouteStatus</code> .

Return Values

Boolean	Returns TRUE if the shapeRouteStatus for the specified shape is successfully set. Returns FALSE otherwise.

Text Display Objects

A text display object lets you make changes in the display characteristics of master labels on an instance-by-instance basis. All master labels are displayed in the instances of that master.

Every instance label (which has changed any, some, or all of the display characteristics from the master label), creates a text display object. This object has information about the display characteristics for that label on that instance. The changed display information is kept in this object. Other characteristics are inherited from the master label. The master label is called the associate and the instance label is called the owner.

The fields outlining the display characteristics in the text display object are the same as in the label: origin, justification, rotation, font style, font height, overbar, and drafting. In addition, the text display object has three more fields: owner, associate, and displayFlags. The displayFlags function (of type `dbDisplayFlags`) indicates which display characteristics in the text display object override the characteristics from the master label.

Text Display Types

The following table shows the enumerated constants of type `dbTextDisplayAssocType` used for the associate (master label) of text display objects. The type of the text display object indicates the associate type of a text display object.

TextDisplay Types

Associate types	Enumerated constants in type <code>dbTextDisplayAssocType</code>
Label	<code>dbcLabelAssocType</code>
Property	<code>dbcPropAssocType</code>
Attribute	<code>dbcAttrAssocType</code>
Parameter	<code>dbcParamAssocType</code>

The Text Display Function

The only function (as opposed to macro) affecting the display of text is `dbTextDisplayId`.

`dbTextDisplayId`

`dbTextDisplayId`

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbCreateTextDisplay(  
    dbTextDisplayRec txtDisplayRecord, dbDisplayFlags displayFlags  
);
```

Creates a text display object with the specified attributes. `txtDisplayRecord` is a text display record containing the initial values for the display characteristics being changed. `txtDisplayRecord` also has fields for the owner and the associate. The full description of the record structure is described in [“Text Display Objects”](#) on page 452.

The `displayFlags` parameter indicates display characteristics inherited from the master and passed as parameters.

Macros to set and reset the various display characteristics on an object of type `dbDisplayFlags` are described later in this chapter. All overriding display characteristics (as indicated by `displayFlags`) must have valid values. If any of the fields in the record have invalid values, the default from the associate is used. If any values need to be changed later, you can use the `set` functions.

The `associate` field in the record refers to the label whose display characteristics are being modified. The `owner` field refers to the instance on which the label’s display is being modified. The `bBox`, `justify`, `fontStyle`, `height`, `orient`, `overbar`, `drafting`, and `origin` fields are the display characteristics you can modify. The `text` field is copied from the associate and cannot be modified. The text display object is created on the same layer with the same purpose as the associate.

Macros for Display Flags

The following macros manipulate `displayFlags` and the `dbDisplayFlags` type.

dbmSetDisplayFlagsAll

```
dbmSetDisplayFlagsAll(  
    displayFlags  
);
```

Sets all flags in `displayFlags`. This macro indicates that all display characteristics are retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsAll

```
dbmUnsetDisplayFlagsAll(  
    displayFlags  
);
```

Resets all flags in `displayFlags`. This macro indicates that all display characteristics are retrieved from the associate.

dbmSetDisplayFlagsOverbar

```
dbmSetDisplayFlagsOverbar(  
    displayFlags  
);
```

Sets the overbar flag in `displayFlags`. This macro indicates that the overbar value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsOverbar

```
dbmUnsetDisplayFlagsOverbar(  
    displayFlags  
);
```

Resets the overbar flag in `displayFlags`. This macro indicates that the overbar value is retrieved from the associate.

dbmSetDisplayFlagsDrafting

```
dbmSetDisplayFlagsDrafting(  
    displayFlags  
);
```

Sets the drafting flag in `displayFlags`. This macro indicates that the drafting value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsDrafting

```
dbmUnsetDisplayFlagsDrafting(  
    displayFlags  
);
```

Resets the drafting flag in `displayFlags`. This macro indicates that the drafting value is retrieved from the associate.

dbmSetDisplayFlagsFontStyle

```
dbmSetDisplayFlagsFontStyle(  
    displayFlags  
);
```

Sets the font style flag in `displayFlags`. This macro indicates that the font style value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsFontStyle

```
dbmUnsetDisplayFlagsFontStyle(  
    displayFlags  
);
```

Resets the font style flag in `displayFlags`. This macro indicates that the font style value is retrieved from the associate.

dbmSetDisplayFlagsHeight

```
dbmSetDisplayFlagsHeight(  
    displayFlags  
);
```

Sets the font height flag in `displayFlags`. This macro indicates that the font height value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsHeight

```
dbmUnsetDisplayFlagsHeight(  
    displayFlags  
);
```

Resets the font height flag in `displayFlags`. This macro indicates that the font height value is retrieved from the associate.

dbmSetDisplayFlagsJustify

```
dbmSetDisplayFlagsJustify(  
    displayFlags  
);
```

Sets the justification flag in `displayFlags`. This macro indicates that the justification value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsJustify

```
dbmUnsetDisplayFlagsJustify(  
    displayFlags  
);
```

Resets the justification flag in `displayFlags`. This macro indicates that the justification value is retrieved from the associate.

dbmSetDisplayFlagsOrient

```
dbmSetDisplayFlagsOrient(  
    displayFlags  
);
```

Sets the orientation flag in `displayFlags`. This macro indicates that the orientation value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsOrient

```
dbmUnsetDisplayFlagsOrient(  
    displayFlags  
);
```

Resets the orientation flag in `displayFlags`. This macro indicates that the orientation value is retrieved from the associate.

dbmSetDisplayFlagsOrigin

```
dbmSetDisplayFlagsOrigin(  
    displayFlags  
);
```

Sets the origin flag in `displayFlags`. This macro indicates that the origin value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsOrigin

```
dbmUnsetDisplayFlagsOrigin(  
    displayFlags  
);
```

Resets the origin flag in `displayFlags`. This macro indicates that the origin value is retrieved from the associate.

dbmSetDisplayFlagsVisible

```
dbmSetDisplayFlagsVisible(  
    displayFlags  
);
```


Sets the visible flag in `displayFlags`. This macro indicates that the visible value is retrieved from the text display object as opposed to the associate.

dbmUnsetDisplayFlagsVisible

```
dbmUnsetDisplayFlagsVisible(  
    displayFlags  
);
```

Resets the visible flag in `displayFlags`. This macro indicates that the visible value is retrieved from the associate.

Structure and Functions for Text Display Records

A text display record contains attributes of a text display. A text member of the text display points to a permanent string that must not be modified.

The following structure represents a text display record:

```
typedef struct dbTextDisplayRec dbTextDisplayRec;  
struct dbTextDisplayRec {  
    dbType                type;  
    dbTextDisplayId       textDisplay;  
    unsigned int          size;  
    dbCellViewId          cellView;  
    dbNetId               net;  
    dbPinId               pin;  
    dbBBox                bBox;  
    dbPurpose              purpose;  
    dbLayer               layer;  
    dbLabelJustify        justify;  
    dbFontStyle            fontStyle;  
    dbOrient              orient;  
    dbTextDisplayAssocType tdType;  
    Boolean               overbar;  
    Boolean               drafting;  
    Boolean               visible;  
    Boolean               isNameVisible;  
    Boolean               isValueVisible;  
    dbPoint               origin;  
    dbDistance            height;  
    String                text;  
    dbId                  associate;  
    dbId                  owner;  
    dbDisplayFlags        dFlags;  
};
```

The following functions manipulate the attributes in the text display record shown.

dbSetTextDisplayOverbar

Boolean

```
dbSetTextDisplayOverbar(  
    dbTextDisplayId textDisplayId, Boolean overbar  
);
```

Sets the overbar of the text display object indicated by `textDisplayId` to `overbar`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbSetTextDisplayFontStyle

Boolean

```
dbSetTextDisplayFontStyle(  
    dbTextDisplayId textDisplayId, dbFontStyle fontStyle  
);
```

Sets the font style of the text display object indicated by `textDisplayId` to the value `fontStyle`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbSetTextDisplayDrafting

Boolean

```
dbSetTextDisplayDrafting(  
    dbTextDisplayId textDisplayId, Boolean drafting  
);
```

Sets the drafting of the text display object indicated by `textDisplayId` to the value `drafting`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbSetTextDisplayOrigin

Boolean

```
dbSetTextDisplayOrigin(  
    dbTextDisplayId textDisplayId, dbPoint origin  
);
```

Sets the origin of the text display object indicated by `textDisplayId` to the value `origin`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbSetTextDisplayJustify

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbSetTextDisplayJustify(  
    dbTextDisplayId textDisplayId, dbLabelJustify justification  
);
```

Sets the justification of the text display object indicated by `textDisplayId` to the value `justification`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbSetTextDisplayOrient

```
Boolean  
dbSetTextDisplayOrient(  
    dbTextDisplayId textDisplayId, dbOrient rotation  
);
```

Sets the orientation of the text display object indicated by `textDisplayId` to the value `orient`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbSetTextDisplayHeight

```
Boolean  
dbSetTextDisplayHeight(  
    dbTextDisplayId textDisplayId, dbDistance fontHeight  
);
```

Sets the height of the text display object indicated by `textDisplayId` to the value `height`. The function returns TRUE if successful. Otherwise, the function returns FALSE.

dbDeleteTextDisplay

```
Boolean  
dbDeleteTextDisplay(  
    dbTextDisplayId textDisplayId  
);
```

Deletes the text display object indicated by `textDisplayId`. Displays the text with the default display characteristics. For other effects of deleting shapes, see [“dbDeleteShape”](#) on page 415.

dbGetTextDisplay

```
Boolean
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetTextDisplay(  
    dbTextDisplayId textDisplayId, String pText, dbLabelJustify pJustification,  
    dbOrient pRotation, dbFontStyle pFontStyle, dbDistance pFontHeight, Boolean  
    pOverbar, Boolean pDrafting  
);
```

Retrieves the attributes from the text display object indicated by `textDisplayId`.

dbGetTextDisplayOrigin

```
dbPoint  
dbGetTextDisplayOrigin(  
    dbTextDisplayId textDisplayId  
);
```

Returns the x and y coordinates of the origin of the `textDisplay` in a `dbPoint` structure.

dbGetTextDisplayLP

```
Boolean  
dbGetTextDisplayLP(  
    dbTextDisplayId textDisplayId, dbLayer pLayer, dbPurpose pPurpose  
);
```

Retrieves the layer and purpose of `textDisplay`.

dbGetTextDisplayBBox

```
dbBBox  
dbGetTextDisplayBBox(  
    dbTextDisplayId textDisplayId  
);
```

Returns a pointer to the bounding box of the `textDisplay`. This `bBox` structure is read only and is overwritten by the next call to `dbGet*BBox`. Returns NULL if there is an error.

dbGetTextDisplayAssociate

```
dbId  
dbGetTextDisplayAssociate(  
    dbTextDisplayId textDisplayId  
);
```

Returns the `dbId` of the associate of the text display object. For now, the associate of the text display object can only be a label. Returns `dbcNullId` if there is an error.

dbGetTextDisplayOwner

`dbId`

```
dbGetTextDisplayOwner(  
    dbTextDisplayId textDisplayId  
);
```

Returns the `dbId` of the owner of the text display object (for now, the owner can only be an instance). Returns `dbcNullId` if there is an error.

dbAssociateHasTextDisplay

`Boolean`

```
dbAssociateHasTextDisplay(  
    dbId associateId  
);
```

Returns `TRUE` if there are any text display objects referring to the database object indicated by `associateId`. The associate can only be a label.

dbObjectHasTextDisplay

`Boolean`

```
dbObjectHasTextDisplay(  
    dbId ownerId  
);
```

Returns `TRUE` if the object indicated by `objectId` owns any text display objects. Otherwise, returns `FALSE`. The `objectId` can only refer to an instance.

dbGetAssociateTextDisplay

`dbTextDisplayId`

```
dbGetAssociateTextDisplay(  
    dbId associateId, dbId ownerId  
);
```

Given an associate and an owner, this macro checks if a text display object exists for the owner referring to the associate. Returns a `textDisplayId` if the text display object exists or a `dbcNullTextDisplayId` if it does not exist.

dbStartGenTextDisplay

```
dbGenStateId  
dbStartGenTextDisplay(  
    dbId owner  
);
```

dbGenTextDisplay

```
Boolean  
dbGenTextDisplay(  
    dbGenStateId stateId, dbTextDisplayId &textDisplayId  
);
```

These functions generate all text displays of the object indicated by `owner`.

dbSetTextDisplayVisible

```
Boolean  
dbSetTextDisplayVisible(  
    dbTextDisplayId textDisplayId, Boolean visible  
);
```

Sets the text display visible or invisible by using the value of `visible` argument.

dbSetTextDisplayNameVisible

```
Boolean  
dbSetTextDisplayNameVisible(  
    dbTextDisplayId textDisplayId, Boolean visible  
);
```

Sets the text display name visible or invisible by using the value of `visible` argument.

dbSetTextDisplayValueVisible

```
Boolean  
dbSetTextDisplayValueVisible(  
    dbTextDisplayId textDisplayId, Boolean visible  
);
```

Sets the text display value visible or invisible by using the value of `visible` argument.

Instance and Hierarchy Functions

This section describes instances and the hierarchy, and the functions used for both. See the following sections for more specific information:

Instances

A cellview can contain an instance, and an instance of a cellview can also contain its own cellview. The cellview referred to by an instance (the cellview containing the instance) is called the master cellview of the instance. Many instances can refer to (that is, be contained by) the same master cellview. The master cellview can in turn contain instances that refer to other masters to form multiple levels of hierarchy.

Each instance must be placed where the origin of the master cellview is aligned when instantiated, in one of eight possible orientations. Each instance has a name that must be unique among all instances contained in the cellview.

There are two types of instances: regular instances (`inst`) and mosaic instances (`mosaicInst`). These two object types form the object class called any instance (`anyInst`).

Iterated Instances

An iterated instance is a vector of single instances (instances with the `numInst` attribute equal to one) represented compactly by one database object. The iterated instance refers to its master multiple times (the number of times specified by `numInst`) inside the cellview containing it. The iterated instance has a `numInst` attribute of greater than one.

Instance Name Syntax

The instance name syntax below is in BNF (Backus-Naur Format) notation. Items enclosed in brackets (`[ ]`) specify the item is optional where zero items or one item might be present. Items enclosed in curly braces (`{ }`) are also optional, except zero or more items can be present. The word `number` indicates a nonnegative integer less than 65536.

```
instName ::= baseName [ < number [ : number ] > ]
```

An instance name (`instName`) is a base name that is optionally followed by a simple vector term. The restrictions on the base name and number are the same as the ones for the net and terminal name syntax with some minor restrictions.

The simple vector term can consist of a single number or imply a range of numbers. The range can be expressed by a starting and ending number separated by colons. The vector

term has the same expanded form as the net and terminal names except that for instance names, increment, repeat operators and 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.

Instance Headers

To operate efficiently and keep good records, the database automatically creates and manages a database object type called instance header (`instHeader`). This header serves as a symbolic link between instances and their masters.

Within a cellview, all instances with the same master are linked together under a single instance header for fast traversal. Each instance header, therefore, corresponds to a unique master instantiated in the cellview. Objects belonging to `instHeader` type cannot have properties attached to them.

The `timeStamp` field in the instance header is updated when any instance linked under an instance header is created, deleted, modified, moved, or copied.

Binding of Instances and Instance Headers

The master of an instance might or might not reside in virtual memory. If the master of an instance is in virtual memory and is linked to an instance header, the instance header (and the instances it represents) is bound. Otherwise, the instance header is unbound.

When an instance header is bound, the master cellview is linked to the instance header through an “inter-cellview” reference. When an instance header is unbound, only the name of the master is available from the instance header.

When an instance header is bound to a master, the connections belonging to each instance (represented by the instance header) are also bound to their corresponding terminals on the master.

When a connection in an instance is bound, the instance is also linked to its corresponding terminal using an inter-cellview reference pointer. If a connection cannot be bound (for example, the corresponding terminal does not exist on the master), then only the name of the terminal is available from the unbound connection. Refer to [“CDBA Connectivity Functions”](#) on page 679 for information about connections.

When combined with instance headers, binding adds a level of memory management between instances and their masters. With these concepts, it is possible for CDBA to read

master cellviews into virtual memory only on demand. In this way, not all the masters referred to by a cellview need to be in virtual memory at once. They are only read in as needed. Each time a master needs to be bound, `dbOpen` is called to open the master cellview.

Automatic Actions on Instance Headers and Cellviews

When the first instance of a master is created within a cellview, CDBA automatically creates a new instance header corresponding to the new master in the cellview. When the last instance of a master within a cellview is deleted, the instance header corresponding to that master is automatically deleted from the cellview.

If a master cellview is purged from virtual memory by a call to `dbPurge`, all references to it are invalidated immediately. The `dbPurge` function handles this automatically by going through all cellviews that refer to the master and unbinding all corresponding instance headers.

Instance Creation and Modification

The following functions create instances or modify instances.

dbCreateInst

```
dbInstId
dbCreateInst (
    dbCellViewId    cellViewId,
    dbCellViewId    masterId,
    String           instName,
    dbPoint          origin,
    dbOrient         orient,
    unsigned int     numInst
);
```

Creates an instance of the master specified by `masterId`. The instance is placed at the specified origin with the specified orientation.

The `instName` parameter specifies the name of the instance, which must be unique within the cellview.

The `numInst` parameter specifies the number of instances represented by this database object. `numInst` must have a value of one unless you want to create an iterated instance.

The number of instances implied by `instName` takes precedence over the number specified by `numInst`. If `instName` is NULL, then CDBA names the instance by appending a number to the string valued property `instNamePrefix`, found on the master. If the property does not exist or is not string valued, it uses the default prefix of "I."

The number is updated each time an automatically named instance is created in the cellview.

The following table lists the eight orientations allowed in the enumerated type `dbOrient`. The orientations are a combination of rotation and mirroring about the x or y axis. Mirroring, if any, is done before the instance is rotated by the specified number of degrees.

Orientations of Instances

Mirroring and rotation	Enumerated constant in type <code>dbOrient</code>
No rotation	<code>dbcR0</code>
Rotate 90 degrees	<code>dbcR90</code>
Rotate 180 degrees	<code>dbcR180</code>

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Orientations of Instances , *continued*

Mirroring and rotation	Enumerated constant in type dbOrient
Rotate 270 degrees	dbcR270
Mirror about the Y axis	dbcMY
Mirror about the Y axis, then rotate 90 degrees	dbcMYR90
Mirror about the X axis	dbcMX
Mirror about the X axis, then rotate 90 degrees	dbcMXR90

dbCreateInstByMasterName

dbInstId

```
dbCreateInstByMasterName (
    dbCellViewId    cvId,
    const char      *libName,
    const char      *cellName,
    const char      *viewName,
    const char      *instName,
    dbPoint         org,
    dbOrient        ornt,
    unsigned int    nInst
);
```

Creates an instance (of cellview libName, cellName, viewName) in the specified the cellview.

dbCreateBlockInst

dbInstId

```
dbCreateBlockInst (
    dbCellViewId    cellViewId,
    dbCellViewId    masterId,
    const char      *instanceName,
    dbPoint         origin,
    dbOrient        orientation,
    unsigned int    numInst,
    Boolean         physOnly
);
```

Creates an instance and specifies its visibility in the module (logical) domain.

The EMH status of the instance is specified through `physOnly`. `TRUE` creates an instance that is physical only. `False` creates an instance that is both logical and physical.

dbCreateBlockInstByMasterName

`dbInstId`

```
dbCreateBlockInstByMasterName(  
    dbCellViewId    cvId,  
    const char      *libName,  
    const char      *cellName,  
    const char      *viewName,  
    const char      *instName,  
    dbPoint         org,  
    dbOrient        ornt,  
    unsigned int    nInst,  
    Boolean         physOnly  
);
```

Creates an instance (of cellview `libName`, `cellName`, `viewName`) and specifies its visibility in the module (logical) domain.

The EMH status of the instance is specified through `physOnly`. `TRUE` creates an instance that is physical only. `False` creates an instance that is both logical and physical.

dbCreateInstWithPurpose

`dbInstId`

```
dbCreateInstWithPurpose(  
    dbCellViewId cellViewId, dbCellViewId masterId, String instName, dbPoint  
    origin, dbOrient orient, unsigned short numInst, dbPurpose purpose  
);
```

Creates an instance like `dbCreateInst` except the instance's purpose is defined using `purpose` instead of being set to `dbcCellPurpose`.

dbCreateParamInst

`dbInstId`

```
dbCreateParamInst(  
    dbCellViewId cellViewId, dbCellViewId masterId, String instName, dbPoint  
    origin, dbOrient orient, unsigned short numInst, unsigned int nParams,  
    dbParamRec paramArray  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Creates a parameterized cell instance using the specified parameters. The arguments of this function are identical to the ones used in `dbCreateInst` except for the last two: `nParams` and `paramArray`.

The `nParams` parameter specifies how many parameters are listed in the parameter array `paramArray`. Each element of the parameter array is a record containing the parameter name, type, and value. Creating parameterized cell instances using this function is more efficient than calling the `dbCreateInst`, `dbReplaceProp` combination.

The following is the description of the parameter argument `dbParamRec`:

```
struct dbParamRec {
    String      propName;
    dbPropType  propType;
    dbPropValue propValue;
};
```

dbSetInstPlacementStatus

Boolean

```
dbSetInstPlacementStatus(
    dbInstId instId, dbPlacementStatus placementStatus
);
```

For placement and routing, you can add the optional placement status attribute to an instance. The following table lists the five status types allowed in the enumerated type `dbPlacementStatus`. The default value of this optional attribute (that is, the value set by `dbCreateInst`) is `dbcFirm`.

Instance Placement Status

Placement status of instance	Enumerated constants in type <code>dbPlacementStatus</code>
Unplaced instance	<code>dbcUnplaced</code>
Unplaced with suggested location	<code>dbcSuggested</code>
Placed instance	<code>dbcPlaced</code>
Placed and locked into position	<code>dbcLocked</code>
Cannot be changed	<code>dbcFirm</code>

dbSetInstName

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbSetInstName(  
    dbInstId instId, String instName  
);
```

Sets the name of the specified instance to `instName`. The name of each instance must be unique within the cellview containing the instance. The name must also agree with the number of iterations on the instance.

dbSetInstOrigin

```
Boolean  
dbSetInstOrigin(  
    dbInstId instId, dbPoint newOrigin  
);
```

Sets the origin of the instance specified by `instId` to `newOrigin`. If the instance is connected to any nets, these connections are preserved.

dbSetInstOrient

```
Boolean  
dbSetInstOrient(  
    dbInstId instId, dbOrient newOrientation  
);
```

Sets the orientation of the instance specified by `instId` to `newOrientation`. If the instance is connected to any nets, these connections are preserved.

dbSetInstMag

```
Boolean  
dbSetInstMag(  
    dbInstId instId, float mag  
);
```

Sets the magnification of the `instId` instance to `mag`.

dbSetInstTransform

```
Boolean  
dbSetInstTransform(  
    dbInstId instId, dbTransform transform  
);
```

Sets the origin, orientation, and magnification of the `instId` instance. You must use this function if origin, orientation, or magnification needs to be set more than once.

dbSetInstMaster

Boolean

```
dbSetInstMaster(  
    dbInstId instId, dbCellViewId masterId  
);
```

Sets the master of the specified instance.

dbSetAnyInstMaster

Boolean

```
dbSetAnyInstMaster(  
    dbAnyInstId anyInstId, dbCellViewId masterId  
);
```

Sets the master of any instance.

Note: It will not change the default params they will remain the same.

dbSetAnyInstNumInst

Boolean

```
dbSetAnyInstNumInst(  
    dbAnyInstId anyInstId, unsigned short numInst  
);
```

Sets the number of instances to be represented by this iterated instance. The name of this instance is changed to reflect the specified number of instances. If the instance name is a vectored name, the lower bound is unchanged. If the name is a scalar, the name is suffixed with a zero-based vector expression. For example:

<code>I<7:3></code>	becomes <code>I<8:3></code>	if <code>numInst</code> is set to 6
<code>I<7:3></code>	becomes <code>I<4:3></code>	if <code>numInst</code> is set to 2
<code>I<7:3></code>	becomes <code>I<7></code>	if <code>numInst</code> is set to 1
<code>I<7></code>	becomes <code>I<7:8></code>	if <code>numInst</code> is set to 2
<code>I1</code>	becomes <code>I<0:7></code>	if <code>numInst</code> is set to 8

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

If the new name overlaps members of other instances, the function returns FALSE and no change is made.

Note: Connections on the instance terminals attached to this instance are not automatically updated.

dbSetAnyInstName

Boolean

```
dbSetAnyInstName (
    dbAnyInstId anyInstId, String instName
);
```

Sets the instance name. The number of instances represented by this instance (`numInst`) is set to the one implied by the new name. Connections on the instance terminals are not automatically updated according to the new `numInst`. The function returns FALSE if the new name is not unique, if there is an illegal iterated instance reference, or if the new name overlaps with members of other instances.

dbSetAnyInstBaseName

Boolean

```
dbSetAnyInstBaseName (
    dbAnyInstId anyInstId, String baseName
);
```

Sets the instance base name. The function returns FALSE if the resulting instance name is not unique, if there is an illegal iterated instance reference, or if the instance base name overlaps with members of other instances, in which case no change is made.

The `dbOpenHier` function is documented in [“CDBA Input and Output Functions”](#) on page 161

dbAllocFlattenOptions

dbFlattenOptions*

```
dbAllocFlattenOptions (
);
```

Allocates memory for the `dbFlattenOptions` pointer and initializes different options with default values.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

None

Return Value

dbFlattenOptions. Pointer to the allocated memory area.
NULL, in case of failure.

dbFreeFlattenOptions

Boolean

```
dbFreeFlattenOptions(  
    dbFlattenOptions    *flattenOptions  
);
```

Deallocates the memory allocated for the `dbFlattenOptions` pointer.

Arguments

flattenOptions Pointer to the memory area to deallocate.

Return Value

Boolean TRUE, if the memory is deallocated.
FALSE, in case of failure.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFlattenInst

Boolean

```
dbFlattenInst(  
    dbId          objId,  
    unsigned int  level,  
    Boolean       flattenPcells,  
    Boolean       preservePins,  
    Boolean       preserveRodObjects,  
    Boolean       delDetachedBlockages,  
    Boolean       preservepinFigs,  
);
```

Flattens instance *objId* up through named levels of hierarchy. If the instance is flattened a value of `TRUE` is returned, if it is not flattened a value of `FALSE` is returned. This API flattens all the vias. Also, `flattenInst` flattens mosaics as well as normal instances.

Arguments

<i>objId</i>	Database ID of the instance to flatten. To flatten a mosaic, specify the <code>mosaicId</code> .
<i>level</i>	Number of levels of hierarchy through which to flatten the instance. Valid Values: an integer between 0 and 32.
<i>flattenPCells</i>	Specifies whether pcells should be flattened. <code>TRUE</code> indicates that pcells should be flattened. Default values: <code>FALSE</code> . Valid Values: <code>TRUE</code> or <code>FALSE</code> .
<i>preservePins</i>	Specifies whether pin information should be preserved when flattening. When set to <code>TRUE</code> , <code>flattenInst</code> preserves the connectivity information, 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. The <code>preservePins</code> is supported for mosaics as well. Default values: <code>FALSE</code> . Valid Values: <code>TRUE</code> or <code>FALSE</code> .
<i>preserveRODobjects</i>	Specifies whether ROD properties should be preserved. Default values: <code>FALSE</code> . Valid Values: <code>TRUE</code> or <code>FALSE</code> .

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

<i>delDetachedBlockages</i>	Specifies whether the detached blockages should be deleted. Default: FALSE. Valid Values: TRUE or FALSE.
<i>preservePinFigs</i>	Specifies whether the pin figures should be preserved. If set to TRUE, the function preserves the pin figures. Default: TRUE. Valid Values: TRUE or FALSE.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFlattenInst2

Boolean

```
dbFlattenInst2(  
    dbId          objId,  
    unsigned int  level,  
    dbFlattenOptions *options  
    dbExcludePCells excludeObj  
);
```

Flattens an instance *objId* up through named levels of hierarchy along with a set of flatten options. `dbFlattenInst2` flattens mosaics as well as normal instances. You need to allocate and free the `dbFlattenOptions` pointer, before and after using this function by using the [dbAllocFlattenOptions](#) and [dbFreeFlattenOptions](#) functions.

Arguments

<i>objId</i>	Database ID of the instance to flatten.
<i>level</i>	Number of levels of hierarchy through which to flatten the instance. Valid Values: an integer between 0 and 32.
<i>options</i>	Specifies a C Struct with the following fields: ■ <code>flattenPCells</code>: Specifies whether Pcells are flattened. Default: FALSE. Valid Values: TRUE or FALSE. ■ <code>preservePins</code>: Specifies whether pin information is preserved. Default: FALSE. Valid Values: TRUE or FALSE. ■ <code>preserveConn</code>: Specifies whether to preserve only nets and not terminals or pins. Default: FALSE. Valid Values: TRUE or FALSE. ■ <code>preserveTermName</code>: Specifies whether terminal name is preserved. Default: FALSE. Valid Values: TRUE or FALSE.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

- **preserveRodObjs:** Specifies whether ROD properties are preserved.
Default: FALSE.
Valid Values: TRUE or FALSE.
- **delDetachedBlockages:** Specifies whether detached blockages are deleted.
Default: FALSE.
Valid Values: TRUE or FALSE.
- **preservePinFigs:** Specifies whether pin figures are preserved.
Default: TRUE.
Valid Values: TRUE or FALSE.
- **flattenVias:** Specifies whether vias are flattened.
Default: TRUE
Valid Values: TRUE or FALSE.
- **hierSeparator:** A string of characters specifying value of hierarchy separator as a string.
Default: none.
- **callbackOnFigCopy:** Callback function pointer (dbOnFigCopyPtr) on Fig copy with the following signature:

```
typedef void (*dbOnFigCopyPtr) (dbFigId srcFig,  
dbFigId dstFig);
```

Example:

```
void FigCallback(dbFigId srcFig,dbFigId dstFig);  
options->callbackOnFigCopy = figCallback;
```

- **callbackOnMosaicInstCopy:** Callback function pointer (dbOnMosaicInstCopyPtr) on Mosaic instance copy with the following signature:

```
typedef void (*dbOnMosaicInstCopyPtr)  
(ddbMosaicId oMosaic, dbInstId nInst, int  
row, int column);
```

Example:

```
void mosaicInstCallback(dbMosaicId oMosaic,  
dbInstId nInst, int row, int column);  
options->callbackOnMosaicInstCopy =  
mosaicInstCallback;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

- **skipSteiners**: When specified, skips Steiners while flattening an instance.
Default: FALSE.
Valid Values: TRUE or FALSE.
- **skipFluidShapes**: When specified, skips fluid shapes while flattening an instance.
Default: FALSE.
Valid Values: TRUE or FALSE.
- **useViaPurpose**: Sets the purpose of the copied shapes to the purpose of the parent via.
Default: FALSE.
Valid Values: TRUE or FALSE.
- **preserveNetsWithNoTerms**: Specifies whether nets that are not connected to any terminal are to be preserved.
Valid Values: TRUE or FALSE.
- **preservePinFigLabels**: Specifies whether pin figure labels are preserved during flattening. If set to FALSE, the labels are not flattened.

Default: TRUE
Valid Values: TRUE or FALSE.

This option will have no effect if option `preservePins` is set to FALSE.

excludeObj

A `dbExcludePCells` object.

When specified, any PCell instance whose master library, cell, and view names are stored in this object, will not be flattened, even when `flattenPCells` option is TRUE.

For more information, see
`dbCreateExcludePCellsFromFile`.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Return Value

Boolean

TRUE, if the instance is successfully flattened.

FALSE, in case of failure.

Example

```
void FigCallback (dbFigId srcFig, dbFigId dstFig);

void mosaicInstCallback(dbMosaicId oMosaic, dbInstId nInst, int row, int column);

dbFlattenOptions      *options = dbAllocFlattenOptions();

options->hierSeparator = strdup("_");
options->callbackOnFigCopy = figCallback;
options->callbackOnMosaicInstCopy = mosaicInstCallback;
options->skipSteiners = TRUE;

dbFlattenInst2(instMid, 1, options);

free(options->hierSeparator);
dbFreeFlattenOptions(options);
```

Flattens the instance `instMid` through one level of the hierarchy by using "_" as hierarchy separator. A Steiner, if any, is copied to the parent cell. Callbacks are set for FigCopy and MosaicInstCopy, which are defined in user's code.

dbCreateExcludePCellsFromFile

```
dbExcludePCells  
dbCreateExcludePCellsFromFile(  
    const char* filePath  
);
```

Description

Creates a `dbExcludePCells` object from the input `ExcludePCells` file.

The `dbExcludePCells` object stores the library, cell, and view names of the Pcell masters. When passed as a flatten option, any Pcell instance whose master library, cell, and view names are stored in this object will not be flattened.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

char filePath*

Path to a file that contains the library, cell, and view name of the Pcells to be excluded during flattening, provided in the following SKILL list format:

```
(t_libName [t_cellName [t_viewName]])
```

Where:

- *t_libName*
Name of a library. This is a mandatory value.
- *t_cellName*
Name of a cell. This value is optional and cannot be specified without *t_libName*.
- *t_viewName*
Name of a view. This value is optional and cannot be specified without *t_cellName*.

The contents of a sample exclude Pcells file are as follows:

```
("libA" "cellA" "viewA")  
("libB" "cellB")  
("libC")
```

Here, the following instances of Pcells are not flatted:

- All instances with master cellview as libA/cellA/viewA.
- All instances for which master is any view in libB/CellB.
- All instances for which master is any cellview in libC.

Return Value

dbExcludePCells

A dbExcludePCell object containing Pcell information

Returns `nullptr` in case of error.

dbExcludePCellsContain

Boolean

```
dbExcludePCellsContain(  
    dbExcludePCells dbPCells,  
    dbCellViewId cvId  
);
```

Description

Checks whether the cellview ID of the specified Pcells is to be excluded during Pcell flattening.

Arguments

<i>dbPCells</i>	Database IDs of the related <u>dbExcludePCells</u> object.
<i>cvId</i>	Database ID of Pcell master to be searched.

Return Value

<i>Boolean</i>	TRUE if the specified Pcells exist in the dbExcludePcells object. FALSE if the specified Pcells do not exist in the dbExcludePcells object.

dbDeleteExcludePCells

```
void  
dbDeleteExcludePCells(  
    dbExcludePCells pcells  
);
```

Description

Deletes the dbExcludePCells object.

Arguments

<i>pcells</i>	The dbExcludePCells to the object to be deleted from memory.

Return Values

None

Instance Deletion

The following function deletes the instance from the cellview that contains it. The corresponding instance header is automatically updated to reflect the change. If this instance is the last instance attached to the instance header, then the instance header is also deleted. If the instance is a parent, then all the descendants are also deleted. For more information on parent-child relationships, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

dbDeleteInst

Boolean

```
dbDeleteInst(  
    dbInstId instId  
);
```

Instance Information Retrieval

The following functions retrieve information about instances, including names, instance headers, and masters.

General Instance Information

The following functions retrieve information about instances in general.

dbGetInst

Boolean

```
dbGetInst(  
    dbInstId instId, String pInstName, dbPoint pOrigin, dbOrient pOrientation,  
    unsigned short pNumInst  
);
```

Returns the name, origin, orientation, and `numInst` attributes of the instance specified by `instId`.

dbGetInstTransform

dbTransform

```
dbGetInstTransform(  
    dbInstId instId  
);
```

Returns the transform (the orientation, origin, and magnification) of the instance.

dbGetInstOrigin

dbPoint

```
dbGetInstOrigin(  
    dbInstId instId  
);
```

Returns the origin of the instance as a point.

dbGetInstOrient

dbOrient

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetInstOrient(  
    dbInstId instId  
);
```

Returns the orientation of the instance. The function supports both regular and mosaic instances.

dbGetInstMag

```
float  
dbGetInstMag(  
    dbInstId instId  
);
```

Retrieves the magnification of the instance.

dbGetInstName

```
String  
dbGetInstName(  
    dbInstId instId  
);
```

Returns the name of the instance as a cellview string.

dbGetInstNumInst

```
unsigned int  
dbGetInstNumInst(  
    dbInstId instId  
);
```

Returns the `numInst` attribute of the instance.

dbGetInstPlacementStatus

```
dbPlacementStatus  
dbGetInstPlacementStatus(  
    dbInstId instId  
);
```

Retrieves the placement status of the instance.

dbGetInstBBox

```
dbBBox *  
dbGetInstBBox(  
    dbInstId instId  
);
```

Retrieves the bounding box of the instance and returns a pointer to temporary storage. The bounding box of an instance is determined by the bounding box of the master cellview, the origin of the instance, its orientation, and its magnification.

dbGetInstLibName

```
String  
dbGetInstLibName(  
    dbInstId instId  
);
```

Retrieves the library name of the master of the instance.

dbGetInstCellName

```
String  
dbGetInstCellName(  
    dbInstId instId  
);
```

Retrieves the cell name of the master of the instance.

dbGetInstViewName

```
String  
dbGetInstViewName(  
    dbInstId instId  
);
```

Retrieves the view name of the master of the instance.

dbGetInstNameNumInst

```
int  
dbGetInstNameNumInst(  
    String instName  
);
```

Returns the number of iterations implied by the instance name.

```
dbCellViewId  
dbGetAnyInstCellView(  
    dbAnyInstId anyInstId  
);
```

Retrieves the parent cellview of the instance.

dbGetAnyInstNumInst

```
int  
dbGetAnyInstNumInst(  
    dbAnyInstId anyInstId  
);
```

Returns the number of instances represented by this iterated instance.

dbGetAnyInstName

```
String  
dbGetAnyInstName(  
    dbAnyInstId anyInstId  
);
```

Retrieves the name of any instance.

dbGetAnyInstBaseName

```
String  
dbGetAnyInstBaseName(  
    dbAnyInstId anyInstId  
);
```

Retrieves the `baseName` of any instance.

Instance Headers

The following functions retrieve information about instance headers.

dbGetInstInstHeader

```
dbInstHeaderId  
dbGetInstInstHeader(  
    dbInstId instId  
);
```

Returns the instance header of the instance.

dbGetInstHeaderMaster

```
dbCellViewId  
dbGetInstHeaderMaster(  
    dbInstHeaderId instHeaderId  
);
```

Returns the master cellview specified by the instance header. If the master cellview is not already in virtual memory, it is brought into virtual memory and bound to the instance header.

dbGetAnyInstInstHeader

```
dbInstHeaderId  
dbGetAnyInstInstHeader(  
    dbAnyInstId anyInstId  
);
```

Returns the instance header of any instance.

Instance Masters

The functions `dbGetInstMaster` and `dbGetAnyInstMaster`, retrieve information about master cellviews of instances, specifically, the master cellview ID specified by the instance. The master cellview is brought into virtual memory (if the master cellview is not already there) and bound to the corresponding instance header.

dbGetInstMaster

```
dbCellViewId  
dbGetInstMaster(  
    dbInstId instId  
);
```

dbGetAnyInstMaster

```
dbCellViewId  
dbGetAnyInstMaster(  
    dbAnyInstId anyInstId  
);
```

dbGetAnyInstSwitchMaster

```
dbCellViewId  
dbGetAnyInstSwitchMaster(  
    dbAnyInstId anyInstId, String viewList  
);
```

This function switches you into a different cellview that has the same library name and cell name as `anyInst`. The `viewList` parameter returns the first available view name of a string of view names you can switch into. `dbGetAnyInstSwitchMaster` returns the sub-master.

If the switched cellview is not a parameterized cell, this function behaves the same way as `dbOpenCellView` but is easier to use. If the switched cellview is a parameterized cell, each property on `anyInst` is checked to see if the property is a parameter of the cellview. However, it does not matter whether the cellview is parameterized or not, the function still returns the sub-master.

For example:

```
dbGetAnyInstSwitchedMaster(anyInstId, "symbolschematic");
```

The `dbOpenAllMaster` function is documented in [“Database I/O Functions”](#) on page 202.

Predicate Functions for Instances

The following functions return Boolean values.

dbIsInstHeaderBound

```
Boolean
dbIsInstHeaderBound(
    dbInstHeaderId instHeaderId
);
```

Returns TRUE if the instance header is bound to its master. Otherwise, the function returns FALSE.

dbIsSubInstHeaderId

```
Boolean
dbIsSubInstHeaderID(
    dbIsSubInstHeaderId instHeaderId
);
```

Returns TRUE if the instHeaderId indicates a subHeader, which means it is the instHeader for a Pcell sub-master. Otherwise, the function returns FALSE. Calling this function does not open the master cellview.

dbIsInstBound

```
Boolean
dbIsInstBound(
    dbInstId instId
);
```

Returns TRUE if the specified instance is bound to its instance header. Otherwise, the function returns FALSE.

dbIsAnyInstBound

```
Boolean
dbIsAnyInstBound(
    dbAnyInstId anyInstId
);
```

Returns TRUE if any instance is bound to its master. Otherwise, it returns FALSE.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsSameMaster

Boolean

```
dbIsSameMaster(  
    dbAnyInstId anyInstId1, dbAnyInstId anyInstId2  
);
```

Returns TRUE if the two instances refer to the same master, that is, if both have the same instance header as returned by `dbGetAnyInstInstHeader`. Otherwise, it returns FALSE.

Generator Functions

The following are pairs of functions that start the generator for and generate instances, instance headers, and masters of a cellview (or generate information about these).

dbStartGenInstHeader

```
dbGenStateId  
dbStartGenInstHeader(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all instance headers in the cellview.

dbGenInstHeader

```
Boolean  
dbGenInstHeader(  
    dbGenStateId pState, dbInstHeaderId pInstHeaderId  
);
```

Generates all instance headers in the cellview.

dbStartGenMaster

```
dbGenStateId  
dbStartGenMaster(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all masters specified by the cellview. If `instHeader` is unbound at the time this generator is called, it becomes bound to its master when the generator searches `instHeader`.

dbGenMaster

```
Boolean  
dbGenMaster(  
    dbGenStateId pState, dbCellViewId pMasterId  
);
```

Generates all masters specified by the cellview. If `instHeader` is unbound at the time this generator is called, it becomes bound to its master when the generator searches `instHeader`.

dbStartGenInstHeaderToAnyInst

```
dbGenStateId  
dbStartGenInstHeaderToAnyInst (  
    dbInstHeaderId instHeaderId  
);
```

Starts the generator to return a type number that indicates whether `anyInstId` is a regular instance (`dbcInstType`) or a mosaic instance (`dbcMosaicInstType`), or that indicates that no more instances are to be generated (`dbcNoMoreObj`, which is equal to 0).

dbGenInstHeaderToAnyInst

```
dbType  
dbGenInstHeaderToAnyInst (  
    dbGenStateId pState, dbAnyInstId pAnyInstId  
);
```

Returns a type number that indicates whether `anyInstId` is a regular instance (`dbcInstType`) or a mosaic instance (`dbcMosaicInstType`), or that indicates that no more instances are to be generated (`dbcNoMoreObj`, which is equal to 0).

You typically use `dbGenInstHeaderToAnyInst` in a conditional expression of a “while” statement or in a selection expression of a “switch” statement.

dbStartGenInstHeaderToInst

```
dbGenStateId  
dbStartGenInstHeaderToInst (  
    dbInstHeaderId instHeaderId  
);
```

Starts the generator for all instances specified by the instance header.

dbGenInstHeaderToInst

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGenInstHeaderToInst(  
    dbGenStateId pState, dbInstId pInstId  
);
```

Generates all instances specified by the instance header.

dbStartGenAnyInst

```
dbGenStateId  
dbStartGenAnyInst(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all instances and mosaic instances contained in the cellview. The order might be random, so not all instances of the same master are generated in sequence.

dbGenAnyInst

```
dbType  
dbGenAnyInst(  
    dbGenStateId pState, dbAnyInstId pAnyInstId  
);
```

Generates all instances and mosaic instances contained in the cellview. The order might be random, so not all instances of the same master are generated in sequence.

`dbGenAnyInst` returns a type number that indicates whether `anyInstId` is a regular instance (`dbcInstType`), a mosaic instance (`dbcMosaicInstType`), or indicates that no more instances are to be generated (`dbcNoMoreObj`, which is equal to 0).

You typically use `dbGenAnyInst` in a conditional expression of a `while` statement or in a selection expression of a `switch` statement.

dbStartGenInst

```
dbGenStateId  
dbStartGenInst(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all instances contained in the cellview.

dbGenInst

Boolean

```
dbGenInst(  
    dbGenStateId pState, dbInstId pInstId  
);
```

Generates all instances contained in the cellview.

dbStartGenMasterToInstHeader

dbGenStateId

```
dbStartGenMasterToInstHeader(  
    dbCellViewId masterId  
);
```

Starts the generator for all instance headers that are bound to the master cellview specified by `masterId`. Each of these instance headers resides within a different cellview.

dbGenMasterToInstHeader

Boolean

```
dbGenMasterToInstHeader(  
    dbGenStateId pState, dbInstHeaderId pInstHeaderId  
);
```

Generates all instance headers that are bound to the master cellview specified by `masterId`. Each of these instance headers resides within a different cellview.

Hierarchy Traversal Functions

The following functions traverse the hierarchy or retrieve information about it.

dbProduceOverlap

Boolean

```
dbProduceOverlap(  
    dbCellViewId  cellViewId, dbBBox  *pBBox, unsigned int  displayStartLevel,  
    unsigned int  displayStopLevel, dbLayer  layer, dbPurpose  purpose,  
    dbShapeConsumerFunc  shapeConsumer, void  *pConsumerData  
);
```

Calls the user function `shapeConsumer` with each shape whose `bBox` overlaps the bounding box passed in when the shape is transformed to the top-level coordinate system. You can limit the selection to supply shapes that are at certain levels in the hierarchy and only on a specified layer or a specified layer-purpose pair.

The functions `displayStartLevel` and `displayStopLevel` specify the highest level (inclusive) and the deepest level (inclusive) to be produced, respectively. If the layer is set to `dbcAllLayer`, shapes are returned from all layers. The shapes are similarly filtered by the purpose specified. You can use the `dbcAllPurpose` value as a wild card.

The `dbProduceOverlap` function normally returns a value of `TRUE`. However, for both `displayStartLevel` and `displayStopLevel`, if the `cellViewId` is invalid, the function prints a message and returns `FALSE`. If the layer-purpose pair does not exist within the cellview, the function does not print a message and returns `FALSE`. The function prints a message if the bounding box is malformed.

If the shape consumer returns `TRUE`, the producer continues to generate shapes. If it returns `FALSE`, the producer stops immediately and returns `TRUE`.

The consumer data is a pointer into a data structure that you define. It is passed down to the consumer with each call and provides a simple mechanism for reentrant programming. The consumer data can be `NULL`.

The shape record contains a copy of the shape's attributes. Geometric attributes, such as point, bounding box, and orientation, are converted to the coordinate system of the top level cellview (`cellViewId`).

usrShapeConsumer

Declare the consumer for `dbProduceOverlap` as follows:

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Boolean

```
usrShapeConsumer(  
    dbCellViewId cellViewId, dbShapeId shapeId, void *pUsrData,  
    dbShapeRec *pShapeRec  
);
```

dbProduceOverlapInst

Boolean

```
dbProduceOverlapInst(  
    dbCellViewId cellViewId, dbBBox *pBBox, unsigned int displayStartLevel,  
    unsigned int displayStopLevel, dbInstConsumerFunc instConsumer,  
    void *pConsumerData  
);
```

The `dbProduceOverlapInst` function provides most of the same information as the `SKILL getOverlaps` function does. Unlike the `SKILL getOverlaps` function, however, the `dbProduceOverlapInst` function does not produce shapes; the `dbProduceOverlapInst` function produces only overlapping instances.

The `dbProduceOverlapInst` function produces instances similar to the way `dbProduceOverlap` produces shapes. The `dbProduceOverlapInst` function returns only the instances between levels `displayStartLevel` and `displayStopLevel`, inclusive, that overlap the bounding box `pBBox` when the instance is transformed to the top-level coordinate system.

The `instConsumer` argument is input to `FuncBooleanPtr`. This is necessary for the code to work with the `c++` compiler. For Example:

```
dbProduceOverlapInst(  
    cv, pbbox, dbcInstanceLayer, dbcInstanceLayer,  
    (FuncBooleanPtr)myInstConsumer, NULL);
```

usrInstConsumer

The instance record contains a copy of the instance's attributes. Geometric attributes, such as origin, bounding box, and orientation, are converted to the coordinate system of the top-level cellview (`cellViewId`). Declare the consumer as follows.

Boolean

```
usrInstConsumer(  
    dbCellViewId, dbInstId, void *, dbInstRec  
);
```

The `dbProduceOverlapInst` function is based on `dbGetNeighbor` and has the same calling conventions and limitations. The function returns only hierarchical instances found in

the direction specified. The `dbProduceOverlapInst` function does not produce vias. Use the `dbRegionQuery` C API function to produce vias.

dbGetNeighbor

`dbShapeId`

```
dbGetNeighbor (
    dbCellViewId    cellViewId
    dbBBox           *pBBox
    dbDirection      direction
    dbLayer          layer
    dbPurpose        purpose
    dbDistance       *pDistance,
    dbPoint          *pXyOffset
);
```

Returns the closest neighboring shape on the specified layer and purpose.



Shapes that overlap the bounding box are ignored.

The `dbGetNeighbor` function and its companion function, `dbGetNeighborInst` (described after this function), provide the same functionality as the SKILL `dbGetNeighbor` function.

The `dbGetNeighbor` function generates only shapes at the current display level in the hierarchy. The cellview passed in must be the cellview being edited within a window on the current desk. If it is not, a message is printed and `dbcNullShapeId` is returned. `dbcNullShapeId` is returned if nothing is found on the layer-purpose pair or if the layer-purpose pair does not exist within the cellview. If the cellview does not exist or the bounding box is not well-formed, a message is printed before returning a `dbcNullShapeId`.

dbGetNeighborInst

`dbInstId`

```
dbGetNeighborInst (
    dbCellViewId    cellViewId
    dbBBox           *pBBox
    dbDirection      direction
    dbDistance       *pDistance
    dbPoint          *pXyOffset
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Returns only the hierarchical instances found in the direction specified. This function is based on `dbGetNeighbor` (previously described) and has the same calling conventions and limitations.

Advanced Hierarchy Traversal Functions

The following structures can be used to retrieve more detailed information about the hierarchy.

dbHierPath Structure

The `dbHierPath` structure contains information about the hierarchy.

Declarations

The declarations for the `dbHierPath` structure are the following.

```
struct dbHierPath {
    dbId          id,
    unsigned int   rows,
    unsigned int   columns,
    unsigned int   memInst,
    dbBBox         bBox
}
typedef struct dbHierPath dbHierPath;
```

Arguments

The members of the `dbHierPath` structure are the following.

<code>id</code>	Maintains a history of all hierarchical references traversed. The hierarchical references can be anyInsts or vias. Set by <code>dbProduceOverlap2</code> .
<code>row</code>	Indicates the row on which <code>mosaicInst</code> exists. Set by <code>dbProduceOverlap2</code> .
<code>columns</code>	Indicates the column on which <code>mosaicInst</code> exists. Set by <code>dbProduceOverlap2</code> .
<code>memInst</code>	Indicates the member instance on which <code>mosaicInst</code> exists. Set by <code>dbProduceOverlap2</code> .

dbHierInfo Structure

The `dbHierInfo` structure contains detailed information about the hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Declarations

The declarations for the `dbHierInfo` structure are the following.

```
struct dbHierInfo {
    dbCellViewId    topCellViewId;
    dbLayer         layer;
    dbPurpose       purpose;
    dbBBox         bBox;
    FuncConsumerPtr usrConsumer;
    usrType         *pCArg;
    FuncSwitcherPtr usrSwitcher;
    usrType         *pSArg;
    dbCellViewId    cellViewId;
    dbId            id;
    int             threshold;
    Boolean         bordersOnly;
    Boolean         toExpand;
    Boolean         toProduce;
    dbTransform     transform;
    int             depth;
    dbHierPath      hierPath[dbcMaxHierDepth];
    FuncSwitcherPtr mosaicSwitcher;
    usrType         *pMArg;
    Boolean         useThresholdOnDot;
    Boolean         doNeighbor;
    dbDirection     direction;
    dbDistance      distance;
    dbPoint         closePoint;
    Boolean         pruneByLayer;
    usrType         pNeighborInfo;
    unsigned short  borderRows;
    unsigned short  borderColumns;
    int             abortCounter;
    int             rectThreshold;
    Boolean         pinOnly;
    Boolean         toCheckAbort;
    Boolean         trueOverlap;
    FuncSwitcherPtr viaSwitcher
    usrType         *pVArg
};
typedef struct dbHierInfo dbHierInfo;
#define dbcMaxHierDepth 32
```

Arguments

The members of the `dbHierInfo` structure are the following.

<code>topCellViewId</code>	Specifies the ID of the top view where the production starts. Must be initialized by the caller.
<code>layer</code>	Produces only the shapes on this layer. Must be initialized by the caller.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

<code>purpose</code>	Produces only the shapes with this purpose. Must be initialized by the caller.
<code>bBox</code>	Produces any shape that overlaps with it. Must be initialized by the caller.
<code>usrConsumer</code>	Invoked when a shape gets produced. Must be initialized by the caller.
<code>pCArg</code>	Must be initialized by the caller.
<code>usrSwitcher</code>	Invoked when it encounters an <code>anyInst</code> . Must be initialized by the caller.
<code>pSArg</code>	Must be initialized by the caller.
<code>cellViewId</code>	Specifies the ID of the view being switched to. This member is updated by <code>usrSwitcher</code> .
<code>id</code>	An <code>anyInst</code> ID can be used by <code>usrSwitcher</code>. A mosaic ID can be used by <code>mosaicSwitcher</code>. A via ID can be used by <code>viaSwitcher</code>. Because it is a valid database object ID, you can use it to do the necessary acquisition. This member is set by <code>dbProduceOverlap2</code>.
<code>threshold</code>	Sets up the minimum size of the bounding box of the shape to produce. Can be initialized by the caller.
<code>bordersOnly</code>	Ignored by <code>OpenAccess</code> .
<code>toExpand</code>	If set to <code>TRUE</code> , makes <code>dbProduceOverlap2</code> continue to traverse down the current hierarchy. This member is updated by <code>usrSwitcher/</code> <code>mosaicSwitcher/</code> <code>viaSwitcher</code> .
<code>toProduce</code>	If set to <code>TRUE</code> , makes <code>dbProduceOverlap2</code> continue to produce the shapes of the current hierarchy. This member is updated by <code>usrSwitcher/</code> <code>mosaicSwitcher/</code> <code>viaSwitcher</code> .
<code>transform</code>	The concatenated transformation to the top-cell coordinate system. This member is updated by <code>dbProduceOverlap2</code> .
<code>depth</code>	Indicates the level of the current hierarchy. This member is updated by <code>dbProduceOverlap2</code> .
<code>hierPath[dbcMaxHierDepth]</code>	

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

	The path to the current object. It is an array with a size of 32. It is set by <code>dbProduceOverlap2</code> and must not be altered by the user.
<code>mosaicSwitcher</code>	Called when a <code>mosaic</code> is encountered. Must be initialized by the caller.
<code>pMArg</code>	Must be initialized by the caller.
<code>useThresholdOnDot</code>	Ignored by <code>OpenAccess</code> .
<code>doNeighbor</code>	Internal. No meaning to users.
<code>direction</code>	Internal. No meaning to users.
<code>distance</code>	Internal. No meaning to users.
<code>closePoint</code>	Internal. No meaning to users.
<code>pruneByLayer</code>	If set to <code>TRUE</code> , skips the switcher when the master of the instance has no layer or purpose specified.
<code>pNeighborInfo</code>	Internal. No meaning to users.
<code>borderRows</code>	Ignored by <code>OpenAccess</code> .
<code>borderColumns</code>	Ignored by <code>OpenAccess</code> .
<code>abortCounter</code>	Defaults to -1 if you signal an abort. Otherwise, it is internal. No meaning to users.
<code>rectThreshold</code>	A shape whose size exceeds the threshold returns as a shape other than a rectangle.
<code>pinOnly</code>	Ignored by <code>OpenAccess</code> .
<code>toCheckAbort</code>	If <code>TRUE</code> , catches an abort signal (^C) and stops traversing the hierarchy. Defaults to <code>FALSE</code> , which ignores an abort signal.
<code>trueOverlap</code>	<code>OpenAccess</code> always produces objects based on true overlaps. Part of a figure must overlap the search box to produce the figure. If the <code>bbox</code> overlaps the search box, but no part of the figure overlaps the search box, no figure is produced.
<code>viaSwitcher</code>	Invoked when a via is encountered. Must be initialized by the caller.
<code>pVArg</code>	Must be initialized by the caller.

Functions for the HierInfo Structure

The following functions allocate and free the `HierInfo` structure.

dbAllocHierInfo

```
dbHierInfo *  
dbAllocHierInfo(  
    );
```

This function allocates and initializes a `dbHierInfo` structure. If the function fails, `NULL` is returned.

dbFreeHierInfo

```
Boolean  
dbFreeHierInfo(  
    dbHierInfo pHierInfo  
    );
```

This function frees the `dbHierInfo` structure. If the function fails, `FALSE` is returned.

dbProduceOverlap2

```
Boolean  
dbProduceOverlap2(  
    dbHierInfo *pHierInfo  
    );
```

Calls the user-defined consumer and user-defined view switcher with the top view ID stored in `pHierInfo->topCellViewId`. When transformed to the top-level coordinate system, each shape is on the layer-purpose pair and overlaps the searching box. The function produces `pHierInfo->bBox` and passes it to the consumer `pHierInfo->usrConsumer`.

For each instance, `dbProduceOverlap2` calls the view switcher `pHierInfo->viewSwitcher`. You can direct `toProduce` and `toExpand` to govern whether to produce the shapes or not and to expand down the hierarchy or not.

Note: `OpenAccess` always produces objects based on true overlaps. It means that if a figure `bbox` overlaps the search area but not its boundary, the figure will not be produced.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Example

The following is an example of the `dbProduceOverlap2` function for producing via shapes:

```
oaBoolean
dbViaShapesTest::testViaShapes(dbId design)
{
    techFileId    techId;
    dbHierInfo    *hierInfo;
    dbLayer       layerList[3];
    techId = techGetCellViewTechFile(design);
    if (techCNullTechFileId == techId) {
        return FALSE;
    }
    hierInfo = dbAllocHierInfo();
    hierInfo->topCellViewId = design;
    // The top cellview, the query uses to produce via shapes
    hierInfo->viaSwitcher = viaSwitcher;
    // The user-defined C function to let the user decide whether or not a via master
    // should be traversed
    hierInfo->usrConsumer = shapeConsumer;
    // The user-defined C function to call for providing the produced shapes to the user
    hierInfo->usrSwitcher = usrSwitcher;
    // The user-defined C function to let the user decide whether or not an instance
    // master should be traversed
    hierInfo->purpose      = dbcAllPurpose;
    hierInfo->pruneByLayer = TRUE;
    hierInfo->bBox         = *dbGetCellViewBBox(design);
    hierInfo->pCArg        = (usrType*) this;
    hierInfo->pSArg        = (usrType*) this;
    hierInfo->pVArg        = (usrType*) this;
    layerList[0] = 2;
    layerList[1] = 4;
    layerList[2] = 3;

    for(int i = 0; i < 3; i++) {
        hierInfo->layer = layerList[i];
        dbProduceOverlap2(hierInfo); //produce shapes for this layer/AnyPurpose
    }
    dbFreeHierInfo(hierInfo);
    return TRUE;
}
```

usrConsumer

Declare the user-defined consumer function as follows.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Boolean

```
usrConsumer(  
    dbHierInfo  *pHierInfo, dbShapeRec  *pShapeRec  
);
```

pShapeRec points to a shape record structure whose coordinates are transformed to the coordinate system of the top view. You can use the type member dbShapeRec to find its type and pass pShapeRec to the corresponding record type. For example:

```
pPolygonRec = (dbPolygonRec *) pShapeRec;
```

pHierInfo points to the dbHierInfo structure, which dbProduceOverlap2 updated.

If the return value is TRUE, then dbProduceOverlap2 continues to produce shapes. If the return value is FALSE, dbProduceOverlap2 stops producing additional shapes for the current level.

usrSwitcher

Declare the user-defined cellview switcher as follows.

Boolean

```
usrSwitcher(  
    dbHierInfo  *pHierInfo  
);
```

The pHierInfo parameter points to the dbHierInfo structure which dbProduceOverlap2 sets and updates.

The first time usrSwitcher runs, pHierInfo->id is type NULL. dbProduceOverlap2 initializes pHierInfo->id from the top level cellview. usrSwitcher evaluates pHierInfo->id. It then sets the toProduce and toExpand parameters to direct dbProduceOverlap2 to produce shapes and to expand the design hierarchy.

Subsequently, dbProduceOverlap2 sets pHierInfo->id to type dbcInstType or dbcMosaicInstType. usrSwitcher evaluates pHierInfo->id. It then sets the toProduce and toExpand parameters to direct dbProduceOverlap2 to produce shapes and to expand the design hierarchy.

mosaicSwitcher

Declare the user-defined cellview switcher for handling mosaics as follows:

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
mosaicSwitcher(  
    dbHierInfo  *pHierInfo  
);
```

The `pHierInfo` parameter points to `dbHierInfo` structure which `dbProduceOverlap2` sets and updates.

You can use the mosaic ID `pHierInfo->id` without defining `mosaicSwitcher`, if you do not want to affect the mosaic itself.

This function produces two possible outcomes:

- If `mosaicSwitcher` does not exist, CDBA processes `mosaicInst` and passes it to `usrSwitcher`.
- If `mosaicSwitcher` exists and you want to produce `mosaicInst`, you need to set both `pHierInfo->toExpand` and `pHierInfo->toProduce` to `TRUE` in `mosaicSwitcher`. Unlike `usrSwitcher`, you do not need to set `pHierInfo->cellViewId`.

viaSwitcher

Declare the user-defined cellview switcher for handling vias as follows:

```
Boolean  
mosaicSwitcher(  
    dbHierInfo  *pHierInfo  
);
```

The `pHierInfo` parameter points to the `dbHierInfo` structure which `dbProduceOverlap2` sets and updates. `pHierInfo->id` is the via Id. You do not need to define `viaSwitcher` if you do not want to do anything on the via itself.

Hierarchy Signal Traversing Functions

The following structures and functions work together to traverse signals in the hierarchy.

dbSigHierPath Structure

The `dbSigHierPath` structure contains information about the instances that get traversed.

Declarations

The declarations for the `dbSigHierPath` structure are shown below.

```
struct dbSigHierPath {
    dbAnyInstId    anyInstId;
    unsigned int   memIndex;
};

typedef struct dbSigHierPath dbSigHierPath;
```

Members

The members of the `dbSigHierPath` structure are used in the following way:

<code>anyInstId</code>	Keeps the history of all <code>anyInsts</code> members that get traversed. This member is set by CDBA.
<code>memIndex</code>	Specifies the member index. This member is set by CDBA.

dbSigHierInfo Structure

The `dbSigHierInfo` structure contains general information about a signal that is going to be traversed.

Declarations

The declarations for the `dbSigHierInfo` structure are shown below.

```
struct dbSigHierInfo {
    FuncBooleanPtr    usrConsumer;
    usrType            *pCArg;
    FuncBooleanPtr    usrSwitcher;
    usrType            *pSArg;
    dbCellViewId      childCellViewId;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbSigId      sigId;
Boolean      toExpand;
Boolean      toProduceMemInst;
int          startLevel;
int          stopLevel;
nt          depth;
dbSigHierPath hierPath[dbcMaxHierDepth];
};
typedef struct dbSigHierInfo dbSigHierInfo;
#define dbcMaxHierDepth 32
```

Members

The members of the `dbSigHierInfo` structure are used in the following way.

<code>usrConsumer</code>	Called when any <code>sigId</code> is produced. Must be initialized by the caller.
<code>pCArg</code>	Specifies a pointer to a user-defined structure. Must be initialized by the caller.
<code>usrSwitcher</code>	Called when CDBA needs to go down to another level in the hierarchy. If the traversal is successful, <code>usrSwitcher</code> updates the <code>toExpand</code> member to TRUE and updates the member <code>childCellViewId</code> with a valid <code>cellViewId</code> . This <code>cellViewId</code> is the ID of the view to be switched to. If the traversal is unsuccessful, the <code>usrSwitcher</code> updates the <code>toExpand</code> member to FALSE and returns to CDBA. If <code>usrSwitcher</code> is NULL, it assumes you only want to traverse the top level. Must be initialized by the caller.
<code>pSArg</code>	Must be initialized by the caller.
<code>childCellViewId</code>	Specifies the ID of the view to be switched to. This member is updated by <code>usrSwitcher</code> .
<code>sigId</code>	Starts the traversal as the top-level <code>sigId</code> . After a new signal is traversed (produced), CDBA updates this member with the new <code>sigId</code> and calls <code>usrConsumer</code> . Must be initialized by the caller.
<code>toExpand</code>	Traverses down the current hierarchy if <code>toExpand</code> is set to TRUE. CDBA automatically updates this member to FALSE if the traversal reaches the <code>stopLevel</code> . This member is updated by <code>usrSwitcher</code> .

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

<code>toProduceMemInst</code>	Produces the signals during traversal if set to TRUE. This member is updated by <code>usrSwitcher</code> .
<code>startLevel</code>	Must be initialized by the caller.
<code>stopLevel</code>	Stops the traversal when this level is reached. Must be initialized by the caller.
<code>depth</code>	Updates <code>depth</code> to tell which level of the hierarchy the current hierarchy is in. CDBA initializes <code>depth</code> to zero at the top level.
<code>hierPath</code>	Keeps the history of the traversing path to the current object. This member is an array with size 20. It is set by <code>dbProduceOverlap2</code> and must not be touched by the user.

Functions Using the `dbSigHierInfo` Structure

The following functions work with the `dbSigHierInfo` structure to traverse a signal through the hierarchy.

`dbTraverseSigHier`

Boolean

```
dbTraverseSigHier(  
    dbSigHierInfo pSigHierInfo  
);
```

Traverses a specified signal through the design hierarchy. You specify the signal to be traversed through `pSigHierInfo`. `pSigHierInfo` is a `dbSigHierInfo` structure that passes information required by this function. The traversal can start and stop at any level you specify in `pSigHierInfo`. Refer to `dbSigHierInfo` for details. This function implements the hierarchy traversal in a way similar to `dbProduceOverlap2`.

The `dbTraverseSigHier` function calls the user-defined consumer `usrConsumer` and view switcher `usrSwitcher`. The `dbTraverseSigHier` function is called for every connection made by this signal in a specified view.

If `usrConsumer` returns FALSE or an error is encountered, this function stops immediately and returns FALSE. Otherwise, `dbTraverseSigHier` returns TRUE.

`usrConsumer`

Code the `usrConsumer` function as follows:

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Boolean

```
usrConsumer(  
    dbSigHierInfo pSigHierInfo  
);
```

The pointer `pSigHierInfo` is a pointer to `dbSigHierInfo`, which was updated by `dbTraverseSigHier`. If the return value is `TRUE`, `dbTraverseSigHier` continues to produce signals. If the return value is `FALSE`, `dbTraverseSigHier` stops producing any more signals.

The view switcher opens the desired views in the design hierarchy where the signal is traversed. If you do not specify the `usrSwitcher`, it assumes only top-level traversing.

usrSwitcher

Code the `usrConsumer` function as follows:

Boolean

```
usrSwitcher(  
    dbSigHierInfo pSigHierInfo  
);
```

`pSigHierInfo` points to the `dbSigHierInfo` structure, which has been updated by `dbTraverseSigHier`. If `pSigHierInfo->usrSwitcher` is `NULL`, the `stopLevel` is assumed to be top level (level 0) and produces only top-level signals.

Set the `toExpand` member of the `pSigHierInfo` structure to `TRUE` if you need to continue the traversal. Set it to `FALSE` otherwise. The `usrSwitcher` function automatically sets `toExpand` to `FALSE` if the `stopLevel` is reached.

Note: You must allocate and free the space of `dbSigHierInfo`.

FigGroup Functions

This section describes functions that affect figGroups objects.

About figGroups

A figGroup is designed to provide a grouping mechanism for geometrical objects with some collective property as boundary, orientation and origin.

The basic use model for figGroups is to take individual figures and group them into one unit, fixing the relative positions of the set. After this, the unit acts as one object which moves all figures at once. Another use model is the ability to query base on the figGroup's bounding box.

Object type

```
typedef dbId    dbFigGroupId;
typedef dbcNullFigGroupId  ((dbFigGroupId) NULL)
```

The following list the types of figGroups.

```
typedef enum dbFigGroupType {
    dbcNoneFigGroupType = 0, (VLE editable. Equiavelent to no type being set.)
    dbcNexgenFigGroupType = 1,
    dbcSyncCloneFigGroupType = 2,
    dbcPlaceAreaFigGroupType = 3,
    dbcRowFigGroupType = 4,
}

#define dbcMaxFigGroupType 4

static CONST constString dbvFigGroupTypeString[] = {
    none,
    analogModule,
    syncClone,
    placeArea,
    row
};
```

dbCreateFigGroup

dbFigGroupId

```
dbCreateFigGroup (
    dbCellViewId      cellViewId
    cppConstString    name
    Boolean            ordered
    dbPoint            origin
    dbOrient           orient
    dbFigGroupType    type = dbcNoneFigGroupType
);
```

Description

Creates a figGroup object in specified cellview with specified name. If ordered is true then members' order is preserved. The origin, orient, and type are specified as well. Returns the dbFigGroupId of the newly created figGroup if the command succeeds.

dbDeleteFigGroup

Boolean

```
dbDeleteFigGroup (
    dbFigGroupId      figGroupId
);
```

Description

Deletes a specified figGroup object. Removes the contents of the figGroup but does not delete the contents. Deletes props and the areaBoundary as well. Returns true if command succeeds, false if it fails.

dbAddFigToFigGroup

Boolean

```
dbAddFigToFigGroup (
    dbFigGroupId      figGroupId
    dbId id
);
```

Description

Adds the dbObject Id to the members list of the specified figGroup. Returns true if the member is added successfully.

dbFindFigGroupByName

```
dbFigGroupId  
  
dbFindFigGroupByName (  
    dbCellViewId      cellViewId  
    cppConstString    name  
);
```

Description

Returns the figGroupId if found. Otherwise returns dbcNullFigGroupId.

dbRemoveFigFromFigGroup

```
Boolean  
  
dbRemoveFigFromFigGroup (  
    dbFigGroupId      figGroupId  
    dbId              id  
);
```

Description

Removes the specified dbobject ID from the members list of the specified figGroup. Returns true if Id is successfully removed or it is not in the figGroup's members list.

dbSetFigGroupOrient

```
Boolean  
  
dbSetFigGroupOrient (  
    dbFigGroupId      figGroupId  
    dbOrient          orient  
);
```

Description

Sets the orientation property of the figGroup to the specified orientation. Returns true if successful.

dbSetFigGroupOrigin

Boolean

```
dbSetFigGroupOrigin(  
    dbFigGroupId      figGroupId  
    dbPoint           origin  
);
```

Description

Sets the origin property of the figGroup to the specified point. Returns true if successful.

dbGetFigGroupOrdered

Boolean

```
dbGetFigGroupOrdered(  
    dbFigGroupId      figId  
);
```

Description

Returns true if the specified figGroup is ordered and false otherwise.

dbSetFigGroupOrdered

Boolean

```
dbSetFigGroupOrdered(  
dbFigGroupId      figId  
Boolean          ordered  
);
```

Description

Sets the value of the ordered property of the specified figGroup to the specified value.
Returns true if successful.

dbGetFigGroupName

String

```
dbGetFigGroupName(  
    dbFigGroupId    figGroupId  
);
```

Description

Retrieves the name of the figGroup object specified. Returns the string if found, or NULL.

dbGetFigGroupOrientation

dbOrient

```
dbGetFigGroupOrientation(  
    dbFigGroupId    figGroupId  
);
```

Description

Returns the orientation of the specified figGroup.

dbGetFigGroupOrigin

dbPoint

```
dbGetFigGroupOrigin(  
    dbFigGroupId    figGroupId  
);
```

Description

Returns origin of the specified figGroup.

dbGetFigGroupType

```
dbGetFigGroupType (
    dbFigGroupId    figGroupId
);
```

Description

Returns the type of the specified figGroup.

dbGetMemberFigGroup

```
dbFigGroupId
dbGetMemberFigGroup (
    dbId    id
);
```

Description

Returns the figGroupId whose member list contains the dbId specified. Otherwise returns dbcNullFigGroupId.

dbStartGenFigGroup

```
dbGenStateId
dbStartGenFigGroup (
    dbCellViewId    cellViewId
);
```

Description

Initializes an iterator for all figGroup objects in a cell view. Returns a stateId for this iterator.

dbGenFigGroup

```
Boolean
dbGenFigGroup (
    dbGenStateId    stateId,
    dbFigGroupId    *figGroupId
);
```

Description

Retrieve the next figGroup Id for the state Id. If the list is exhausted, return false. Otherwise returns true.

dbStartGenFigGroupTold

```
dbGenStateId  
dbStartGenFigGroupToId(  
    dbFigGroupId      FigGroupId  
);
```

Description

Initializes an iterator over the list of dbld's of members of the specified figGroup. Returns a stateId for this iterator.

dbGenFigGroupTold

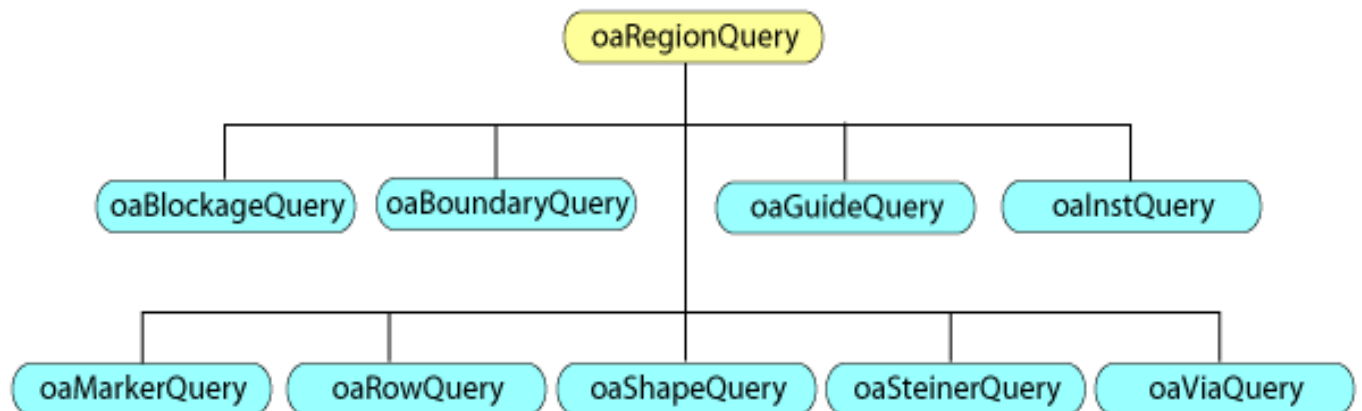
```
Boolean  
dbGenFigGroupToId(  
    dbGenStateId      stateId  
    dbId              *id  
);
```

Description

Retrieve the next member dbld. This could be either a figure or a figGroup. If the list is exhausted, return false. Otherwise returns true.

Region Queries

This section describes region query functions. Below is a simple hierarchical design representation from a query perspective.



There is a query function for each region query class type.

- blockage
- boundaries
- guides
- instances
- markers
- rows
- shapes
- steiners
- vias

Each query functions takes the following common arguments.

- top cellview used as the root of the design hierarchy.
- bounding box specifying the region of the design in the top cellview to perform the query.
- transform applied to the design.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

- size filter resolution.
- start and stop levels of the design to query.
- client data used by the client application.
- client consumer function called for each object produced by the region query.
- client pre-hierarchy expansion function called during the region query before processing either an instance or a via.

The pre-hierarchy expansion functions allows a client application to control what portion of the design hierarchy is processed. If the function returns TRUE, the design hierarchy referenced by the specified instance or via is processed. If the function returns FALSE, the design hierarchy is not processed. If pre-hierarchy expansion is called, the client application defined function receives the dbId of an instance or a via, and row and column information if the dbId is an array instance.

- client post-hierarchy expansion function called only if the pre-hierarchy expansion function is called and returns TRUE.

Post-hierarchy expansion is called only after all the hierarchical design content is produced. If post-hierarchy expansion is called, the client application defined function receives the dbId of an instance or a via, and row and column information if the dbId is an array instance.

The client consumer, pre and post hierarchy expansion functions are passed a dbQueryStateId used to access the region query attributes. The top cellview, current transform, size filter, and client data are examples of the attributes. The following are the function prototypes for the object specific query consumer functions.

```
typedef void (*dbBlockageQueryConsumer) (dbBlockageId    blockageId,
                                         dbQueryStateId  stateId) ;

typedef void (*dbBoundaryQueryConsumer) (dbBoundaryId    boundaryId,
                                         dbQueryStateId  stateId) ;

typedef void (*dbGuideQueryConsumer) (dbGuideId         guideId,
                                       dbQueryStateId    stateId) ;

typedef void (*dbInstQueryConsumer) (dbInstId          instId,
                                      dbQueryStateId    stateId) ;

typedef void (*dbMarkerQueryConsumer) (dbMarkerId       markerId,
                                       dbQueryStateId    stateId) ;

typedef void (*dbRowQueryConsumer) (dbRowId            rowId,
                                    dbQueryStateId    stateId) ;

typedef void (*dbShapeQueryConsumer) (dbShapeId        shapeId,
                                       dbQueryStateId    stateId) ;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
typedef void (*dbSteinerQueryConsumer) (dbSteinerId      steinerId,  
                                         dbQueryStateId  stateId) ;  
  
typedef void (*dbViaQueryConsumer)      (dbViaId        viaId,  
                                         dbQueryStateId  stateId) ;
```

The following are the function prototypes for the pre and post expansion functions.

```
typedef Boolean (*dbPreQueryExpand)      (dbId            id,  
                                         unsigned int    row,  
                                         unsigned int    col,  
                                         dbQueryStateId stateId) ;  
  
typedef Boolean (*dbPostQueryExpand)     (dbId            id,  
                                         unsigned int    row,  
                                         unsigned int    col,  
                                         dbQueryStateId stateId) ;
```

The following are the region query functions.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbLayerBlockageQuery

```
void
dbLayerBlockageQuery(
    dbCellViewId      cellViewId,
    dbLayer            layer,
    dbBBox             *region,
    dbTransform        transform,
    unsigned int        filterSize,
    unsigned int        startLevel,
    unsigned int        stopLevel,
    dbBlockageQueryConsumer blockageConsumer,
    dbPreQueryExpand    preExpander,
    dbPostQueryExpand    postExpander,
    void               *clientData);
```

Description

Begins a region query for blockages on the specified layer in the specified region of the design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>layer</i>	Layer number for the blockage query. Pass <code>dbcAllLayer</code> as the layer number value to produce blockages on any layer.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Blockage filter size. The query produces only blockages with a size larger than the <code>filterSize</code> .
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>blockageConsumer</i>	Call this client application defined function for each blockage produced during the region query. If called, the blockage consumer function receives: ■ the <code>dbBlockageId</code> of the blockage produced ■ the <code>dbQueryStatId</code>.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

`void`

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbPlacementBlockageQuery

void

```
dbPlacementBlockageQuery(  
    dbCellViewId          cellViewId,  
    dbLayer                layer,  
    dbBBox                 *region,  
    dbTransform            transform,  
    unsigned int            filterSize,  
    unsigned int            startLevel,  
    unsigned int            stopLevel,  
    dbBlockageQueryConsumer blockageConsumer,  
    dbPreQueryExpand        preExpander,  
    dbPostQueryExpand        postExpander,  
    void                    *clientData);
```

Description

Begins a region query for placement blockages in the specified region of the design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>layer</i>	Layer number for the blockage query. Pass <code>dbcAllLayer</code> as the layer number value to produce blockages on any layer.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Blockage filter size. The query produces only blockages with a size larger than the <code>filterSize</code> .
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>blockageConsumer</i>	Call this client application defined function for each blockage produced during the region query. If called, the blockage consumer function receives: ■ the <code>dbBlockageId</code> of the blockage produced ■ the <code>dbQueryStatId</code>.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

`void`

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbBoundaryQuery

void

```
dbBoundaryQuery(  
    dbCellViewId          cellViewId,  
    dbBBox                *region,  
    dbTransform            transform,  
    unsigned int           filterSize,  
    unsigned int           startLevel,  
    unsigned int           stopLevel,  
    dbBoundaryQueryConsumer boundaryConsumer,  
    dbPreQueryExpand       preExpander,  
    dbPostQueryExpand      postExpander,  
    void                   *clientData);
```

Description

Begins a region query for boundaries in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Boundary filter size. The query produces only boundaries with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>boundaryConsumer</i>	Call this client application defined function for each boundary produced during the region query. If called, the boundary consumer function receives: ■ the dbBoundaryId of the boundary produced ■ the dbQueryStateId.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGuideQuery

```
void
dbGuideQuery(
    dbCellViewId      cellViewId,
    dbBBox             *region,
    dbTransform        transform,
    unsigned int       filterSize,
    unsigned int       startLevel,
    unsigned int       stopLevel,
    dbGuideQueryConsumer guideConsumer,
    dbPreQueryExpand   preExpander,
    dbPostQueryExpand  postExpander,
    void               *clientData);
```

Description

Begins a region query for guides in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Guide filter size. The query produces only guides with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>guideConsumer</i>	Call this client application defined function for each guide produced during the query. If called, the guide consumer function receives: ■ the dbGuideld of the guide produced ■ the dbQueryStateld.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbInstQuery

void

```
dbInstQuery(  
    dbCellViewId          cellViewId,  
    dbBBox                *region,  
    dbTransform            transform,  
    unsigned int           filterSize,  
    unsigned int           startLevel,  
    unsigned int           stopLevel,  
    dbInstQueryConsumer    instConsumer,  
    dbPreQueryExpand        preExpander,  
    dbPostQueryExpand       postExpander,  
    void                   *clientData);
```

Description

Begins a region query for instances in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Instance filter size. The query produces only instances with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>instConsumer</i>	Call this client application defined function for each instance produced during the query. If called, the instance consumer function receives: ■ the dbInstId of the instance produced ■ the dbQueryStateId.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbMarkerQuery

```
void
dbMarkerQuery (
    dbCellViewId          cellViewId,
    dbBBox                *region,
    dbTransform            transform,
    unsigned int           filterSize,
    unsigned int           startLevel,
    unsigned int           stopLevel,
    dbMarkerQueryConsumer markerConsumer,
    dbPreQueryExpand       preExpander,
    dbPostQueryExpand      postExpander,
    void                  *clientData);
```

Description

Begins a region query for markers in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Marker filter size. The query produces only markers with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>markerConsumer</i>	Call this client application defined function for each marker produced during the query. If called, the marker consumer function receives: ■ the dbMarkerId of the marker produced ■ the dbQueryStatId.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbRowQuery

void

```
dbRowQuery(  
    dbCellViewId          cellViewId,  
    dbBBox                *region,  
    dbTransform            transform,  
    unsigned int           filterSize,  
    unsigned int           startLevel,  
    unsigned int           stopLevel,  
    dbRowQueryConsumer     rowConsumer,  
    dbPreQueryExpand       preExpander,  
    dbPostQueryExpand      postExpander,  
    void                   *clientData);
```

Description

Begins a region query for rows in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Row filter size. The query produces only rows with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>rowConsumer</i>	Call this client application defined function for each row produced during the query. If called, the row consumer function receives: ■ the dbRowId of the row produced ■ the dbQueryStateId.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbShapeQuery

```
void
dbShapeQuery(
    dbCellViewId      cellViewId,
    dbBBox             *region,
    dbTransform        transform,
    unsigned int       filterSize,
    unsigned int       startLevel,
    unsigned int       stopLevel,
    dbShapeQueryConsumer shapeConsumer,
    dbPreQueryExpand   preExpander,
    dbPostQueryExpand  postExpander,
    void               *clientData);
```

Description

Begins a region query for shapes in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Shape filter size. The query produces only shapes with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>shapeConsumer</i>	Call this client application defined function for each shape produced during the query. If called, the shape consumer function receives: ■ the dbShapelId of the shape produced ■ the dbQueryStatelId.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSteinerQuery

```
void
dbSteinerQuery(
    dbCellViewId      cellViewId,
    dbBBox             *region,
    dbTransform        transform,
    unsigned int       filterSize,
    unsigned int       startLevel,
    unsigned int       stopLevel,
    dbSteinerQueryConsumer steinerConsumer,
    dbPreQueryExpand   preExpander,
    dbPostQueryExpand  postExpander,
    void               *clientData);
```

Description

Begins a region query for steiners in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Steiner filter size. The query produces only steiners with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>steinerConsumer</i>	Call this client application defined function for each steiner produced during the query. If called, the steiner consumer function receives: ■ the dbSteinerId of the steiner produced ■ the dbQueryStateId.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbViaQuery

void

```
dbViaQuery(  
    dbCellViewId          cellViewId,  
    dbBBox                *region,  
    dbTransform            transform,  
    unsigned int           filterSize,  
    unsigned int           startLevel,  
    unsigned int           stopLevel,  
    dbViaQueryConsumer    viaConsumer,  
    dbPreQueryExpand       preExpander,  
    dbPostQueryExpand      postExpander,  
    void                  *clientData);
```

Description

Begins a region query for vias in the specified region of the specified design hierarchy.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cellViewId</i>	Root of the design hierarchy for the query to traverse.
<i>*region</i>	Pointer to the bounding box specifying the region of the design relative to the top cellview to use as the basis of the query.
<i>transform</i>	Initial transform to use.
<i>filterSize</i>	Via filter size. The query produces only vias with a size larger than the filterSize.
<i>startLevel</i>	Level of the design hierarchy at which to start the query.
<i>stopLevel</i>	Level of the design hierarchy at which to start the query.
<i>viaConsumer</i>	Call this client application defined function for each via produced during the query. If called, the via consumer function receives: ■ the dbViald of the via produced ■ the dbQueryStateld.
<i>preExpander</i>	Client application defined pre-hierarchy expansion function. Optional. Pass a NULL value if no pre design expansion function is provided.
<i>postExpander</i>	Client application defined post-hierarchy expansion function. Optional. Pass a NULL value if no post design expansion function is provided.
<i>*clientData</i>	Pointer to a client data structure. Consumer, pre and post-hierarchy expansion functions can access and modify the data structure.

Return Values

void

dbAbortQuery

```
void  
dbAbortQuery(  
    dbQueryStateId    stateId  
);
```

Description

Aborts the region query in a user-defined consumer or visitor function while it is executing. An abort marks the query as aborted, but does not indicate an immediate termination of the query.

Depending on the current state of the query and the control point being processed at some level of the design hierarchy, the query aborts systematically. Once the abort is called, the user-defined consumers/visitors are not called.

Arguments

<i>stateId</i>	The region query state ID.

Return Values

```
void
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetQueryLayer

```
dbLayer
dbGetQueryLayer (
    dbQueryStateId    stateId
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to get the layer number. This function applies only to blockage, shape, and steiner queries.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

<i>dbLayer</i>	Returns the layer number defined by the client.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetQueryPurpose

```
dbPurpose  
dbGetQueryPurpose (  
    dbQueryStateId    stateId  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to get the purpose. This function applies only during shape queries.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

dbPurpose	Returns the purpose number defined by the client.

dbSetQueryClientData

Boolean

```
dbSetQueryClientData(  
    dbQueryStateId    stateId,  
    void              *clientData  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to set the region query client data. It is the responsibility of the client to manage this data.

Arguments

<i>stateId</i>	The region query state I.
<i>*clientData</i>	Pointer to the client data, typecast to (<i>void*</i>).

Return Values

Boolean	Returns TRUE if the region query client data is successfully set. Returns FALSE otherwise.

dbGetQueryClientData

```
void*  
dbGetQueryClientData(  
    dbQueryStateId    stateId  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to get the region query client data. It is the responsibility of the client to manage this data.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

<i>void*</i>	Returns a pointer to the client data. The client can typecast the data back to the original data type.

dbGetQueryTopCellView

```
dbCellViewId  
dbGetQueryTopCellView(  
    dbQueryStateId    stateId  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to access the top cellview for which the query was executed.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

<i>dbCellViewId</i>	Returns the top cellview database Id.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetQueryRegion

```
dbBBox*  
dbGetQueryRegion(  
    dbQueryStateId    stateId  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to access the original query region for which the query was executed.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

dbBBox*	Pointer to the query region defined by the client.

dbGetQueryCurrentTransform

```
dbTransform  
dbGetQueryCurrentTransform(  
    dbQueryStateId    stateId  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to access the current transform based on the current level in the hierarchy.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

<i>dbTransform</i>	Returns the transform defined by the client.

dbGetQueryCurrentRegion

```
dbBBox*  
dbGetQueryCurrentRegion(  
    dbQueryStateId    stateId  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to access the current query region at different hierarchy levels as the query traverses through the design hierarchy.

Arguments

<i>stateId</i>	The region query state Id.

Return Values

<i>dbBBox*</i>	Returns a pointer to the current query region.

dbCreateQueryHierPath

dbQueryHierPath*

dbCreateQueryHierPath();

Description

Creates an empty hierarchical path data structure. Hierarchical path functions from client application defined region query consumer and hierarchy expansion functions use the hierarchical path data structure.

Return Values

dbQueryHierPath*	Returns a pointer to the newly created hierarchical path data structure.

dbFreeQueryHierPath

```
void  
dbFreeQueryHierPath(  
    dbQueryHierPath    *hierPath  
);
```

Description

Frees hierarchical path data created by dbCreateQueryHierPath().

Argument

<code>*hierPath</code>	Pointer to the hierarchical path data.

Return Values

`void`

dbGetQueryHierPath

```
void  
dbGetQueryHierPath(  
    dbQueryStateId      stateId,  
    dbQueryHierPath     **hierPath  
);
```

Description

Allows the client application defined consumer and hierarchy expansion functions to access the current query hierPath as the query traverses through the design hierarchy.

Arguments

<i>stateId</i>	The region query state Id.
<i>**hierPath</i>	Pointer to the hierarchical path data defined by the client.

Return Values

```
void
```

dbCopyQueryHierPath

```
dbQueryHierPath*  
dbCopyQueryHierPath(  
    dbQueryHierPath    *hierPath  
);
```

Description

Creates an exact copy of the passed in `hierPath` object.

Arguments

<i>*hierPath</i>	Pointer to the original hierarchical path data object.

Return Values

<code>dbRegionQueryHierPath*</code>	A pointer to the duplicate hierarchical path object.

dbCompareQueryHierPaths

Boolean

```
dbCompareQueryHierPaths (
    dbQueryHierPath*    *hierPath1,
    dbQueryHierPath*    *hierPath2
);
```

Description

Compares two hierPath objects.

Arguments

<i>*hierPath1</i>	Pointer to the first hierarchical path.
<i>*hierPath2</i>	Pointer to the second hierarchical path.

Return Values

Boolean	Returns TRUE if hierPath1 and hierPath2 are same. Returns FALSE otherwise.

dbResetQueryHierPath

```
void  
dbResetQueryHierPath(  
    dbQueryHierPath    *hierPath  
);
```

Description

Resets the hierPath by setting its depth to zero.

Arguments

<i>*hierPath</i>	Pointer to the hierarchical path object.

Return Values

```
void
```

dbGetQueryHierPathTransform

```
dbTransform  
dbGetQueryHierPathTransform(  
    dbQueryHierPath    *hierPath  
);
```

Description

Gets the hierPath context transform by concatenating the transforms of all the instances in the hierPath starting from the lowest instance or via in the hierPath and working up towards level 0.

Arguments

<i>*hierPath</i>	Pointer to the hierarchical path object.

Return Values

dbTransform	Returns the current transform.

dbGetQueryHierPathDepth

```
unsigned int  
dbGetQueryHierPathDepth(  
    dbQueryHierPath    *hierPath  
);
```

Description

Returns the value of the depth parameter from the hierarchical path data structure.

Arguments

<i>hierPath</i>	Pointer to the hierarchical path object.

Return Values

<i>unsigned int</i>	Returns the current depth of the hierPath.

dbGetQueryHierPathLevel

```
void
dbGetQueryHierPathLevel (
    dbQueryHierPath    *hierPath,
    unsigned int        level,
    dbId                *id,
    unsigned int*        row,
    unsigned int*        col
);
```

Description

Gets the instId or the viald at the specified level. If this inst is an arrayInst, the function also returns the row and column information.

Arguments

<i>*hierPath</i>	Pointer to the hierarchical path object.
<i>level</i>	The level at which to get the instId or viald. The specified level should be less than the depth of the hierPath.
<i>*Id</i>	Pointer to the database ID of the instance or via to populate.
<i>row</i>	If the ID is an arrayInst, a pointer to the unsigned integer () to populate as the row value.
<i>col</i>	If the ID is an arrayInst, a pointer to the unsigned integer to populate as the column value.

Return Values

void

dbPushQueryHierPathLevel

```
void
dbPushQueryHierPathLevel (
    dbQueryHierPath    *hierPath,
    dbId                id,
    unsigned int        row,
    unsigned int        col
);
```

Description

Pushes an instId or viald on the hierPath. Clients can provide row and column information for array instances. This adds a new level to the hierPath and the depth increments by one.

Arguments

<i>*hierPath</i>	Pointer to the hierarchical path object.
<i>id</i>	The database ID of the instance or via.
<i>row</i>	The row if the inst is an arrayInst, 0 otherwise.
<i>col</i>	The column if the inst is an arrayInst, 0 otherwise.

Return Values

Boolean	Returns TRUE if the instId is successfully pushed on the hierPath. Returns FALSE otherwise.

dbPopQueryHierPath

```
void  
dbPopQueryHierPath(  
    dbQueryHierPath    *hierPath  
);
```

Description

Pops the hierPath up one level. This reduces a level in the hierPath and the depth decrements by one.

Arguments

<i>*hierPath</i>	Pointer to the hierarchical path object.

Return Values

```
void
```

Parameterized Cells

A parameterized cell is a special cellview that creates variations of itself at run time based on different values of parameters. A special SKILL program stored with the cellview applies the parameters. The parameters are specified as a hierarchical property list on the cellview of the parameterized cell. This list lets you choose the names, default values, and value types for the parameters.

When an instance of a parameterized cell is placed in another cellview, any properties on this instance override the default parameters stored on the master. If no variant of the cellview was created for this set of parameters, the SKILL program creates a run-time variant of the parameterized cell. These variants are not stored on disk. Instead, they are re-created each time the parameterized cell is opened or bound.

Parameterized Cells and Database Organization

Only the `superMaster` cellview defining a parameterized cell is permanently stored in the database. The run-time variants created for different parameter bindings are temporary cellviews that exist for the lifetime of the process or the parameter bindings.

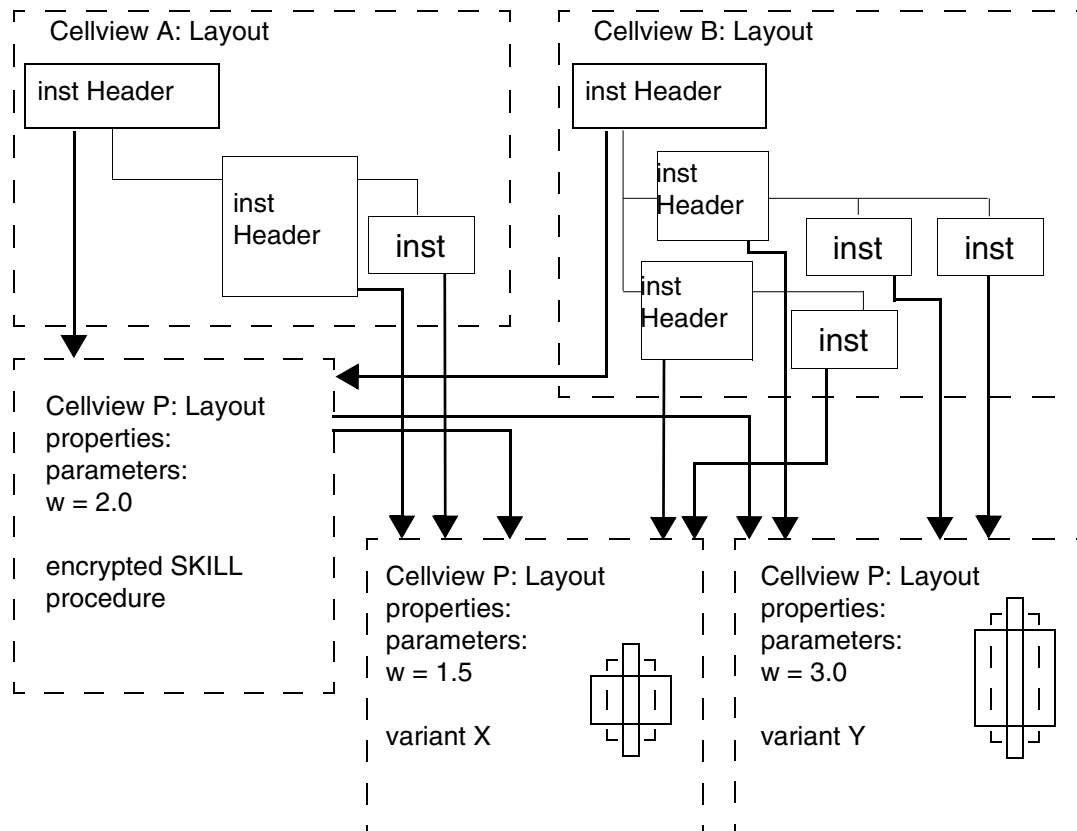
The `dbSave` command does not save a variant cellview because this overwrites the `superMaster` of the parameterized cell. (However, a variant cellview can be written to another cellview using `dbWriteCellView`).

For each unique parameterization of a parameterized cell within a cellview, a variant of `instHeader` describes those instances with that particular parameterization. The instances generated for an `instHeader` include the instances of the variants of an `instHeader` for that `instHeader`.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

In the following figure, cellview P was instantiated once in cellview A and three times in cellview B. The instance in cellview A and one of the instances in cellview B have a property $w=1.5$, while the other two instances in cellview B have a property $w=3.0$.



Functions for Parameterized Cells

The following functions return information and act on cellviews for parameterized cells.

dbCellViewHasVariant

Boolean

```
dbCellViewHasVariant(  
    dbCellViewId cellViewId  
);
```

Returns TRUE if the cellview has variant cellviews.

dbStartGenVariantCellView

```
dbGenStateId  
dbStartGenVariantCellView(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all variant cellviews of a `superMaster` cellview. If the cellview is not a `superMaster`, the function does not generate any cellviews.

dbGenVariantCellView

```
Boolean  
dbGenVariantCellView(  
    dbGenStateId pState, dbCellViewId pCellViewId  
);
```

These two functions generate all variant cellviews of a `superMaster` cellview. If the cellview is not a `superMaster` cellview, the function does not generate any cellviews.

dbInstHeaderHasVariant

```
Boolean  
dbInstHeaderHasVariant(  
    dbInstHeaderId instHeaderId  
);
```

Returns TRUE if the `instHeader` has variant `instHeaders`.

dbStartGenInstHeaderVariant

```
dbGenStateId  
dbStartGenInstHeaderVariant(  
    dbInstHeaderId instHeaderId  
);
```

Prepares the generator for all variant `instHeaders` of an `instHeader`. If the master for `instHeader` is not a parameterized cell, the function does not generate any variant `instHeaders`.

dbGenInstHeaderVariant

```
Boolean
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGenInstHeaderVariant(  
    dbGenStateId pState, dbInstHeaderId pInstHeaderId  
);
```

Generates all variant `instHeaders` of an `instHeader`. If the master of an `instHeader` is not a parameterized cell, the function does not generate any variant `instHeaders`.

dbDumpPcell

Boolean

```
dbDumpPcell(  
    dbCellViewId    cvId,  
    string           cvFile,  
    string           pCellFile  
):
```

Description

Writes out the parameterized cell SKILL procedure attached to the given cellview to the specified file. Also creates a file that contains all the SKILL commands needed to recreate the cellview.

Note: Dumping parameterized cell devices defined in a technology file is not supported in this function.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cvId</i>	The <code>Id</code> of the cellview to be dumped. Must be a parameterized cell.
<i>cvFile</i>	Output file name for the file that can be used to recreate the given cellview.
<i>pCellFile</i>	Output file name for the parameterized cell SKILL procedure.

Return Values

Boolean	Returns TRUE if the parameterized cell SKILL procedure attached to the given cellview to the specified file. Otherwise FALSE is returned.

Defining and Undefined a Parameterized Cell

Described in this section are functions that both convert a regular cellview into a parameterized cellview and a parameterized cellview into a regular cellview. Defining or undefining parameterized cells causes major changes in the database.

When a parameterized cell is created, all `instHeaders` referring to that cellview get reorganized to have variant `instHeaders`. The opposite happens when a parameterized cell is de-parameterized. Defining and undefining can result in opening or purging variants, respectively. Therefore, these functions might run slowly.

dbDefineParamCell

Boolean

```
dbDefineParamCell(  
    dbCellViewId cellViewId, String procFileName  
);
```

Changes the cellview specified by `cellViewId` into a parameterized cell. The procedure filename must be the name of a file containing un-encrypted SKILL code defining the procedure `pcGenCell`.

The cellview must have a hierarchical property named `parameters`. The hierarchical property contains the default values for the parameters. This command does an implicit `dbSave`. If there is a severe error in the SKILL procedure, if the `parameters` property is illegal, or if the function is unable to save the cellview, the function returns FALSE.

dbUndefineParamCell

Boolean

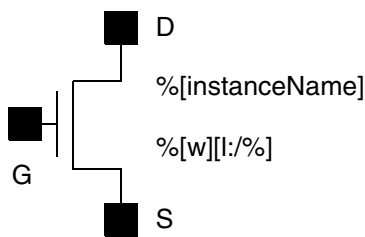
```
dbUndefineParamCell(  
    dbCellViewId cellViewId  
);
```

Changes the parameterized cell specified by `cellViewId` into a regular cellview. The hierarchical property list named `parameters` is deleted from the cellview.

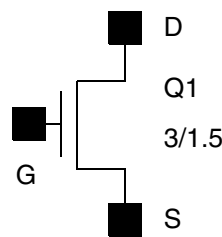
Creating Parameterized Cells

In this section, the process of defining a transistor block described both shows the usefulness of parameterized cells and gives you an example of how to create them.

When you place a transistor symbol, interpreted labels in the transistor symbol usually display values for `w` and `l`. This is seen in the following example of both the definition and an instance of a schematic symbol for an `nfet`.



nfet symbol current



Instance of nfet symbol current
with properties `w=3` and `l=1.5`

Because the netlister is set up to interpret certain properties, it substitutes the correct value for `w` and `l` in the netlist.

However, the process of interpreting properties at run time for the layout representation is more complicated for a netlister. This involves writing the bounding box of a rectangle and the vertices of polygons and paths as an expression that gets interpreted each time the rectangle is referred to. The context for each reference to a rectangle also needs to be passed so that the process can locate the properties.

Parameterized cells evaluate the parameters and compute the effect on all objects in the cellview each time a new set of parameter bindings is encountered. The parameterized cells then create a special temporary cellview, known as a variant cellview. The SKILL procedure

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

associated with the parameterized cell is processed to interpret the parameters and adjust the contents of variant cellview.

To define the transistor layout as a parameterized cell, do the following:

1. In the Graphics Editor, create a hierarchical property list named `parameters`. In this property list, define the parameters and their defaults.

```
GraphicsEditor edit nfet layout
GraphicsEditor plist parameters=proplist(w=2.0 l=1.2)
```

2. Write a SKILL procedure named `pcGenCell` to create the gate and the diffusion, and to implant shapes for the transistors. This procedure is temporarily written to a file, `/tmp/nfet.il`, to be used in the next step:

```
procedure( pcGenCell( cellView )
  prog( (params w l)
    if( !dbobjectp(params = cellView~>parameters)
      error("invalid parameters" ) )
    if( !numberp(w = params~>w) error( "invalid w" ) )
    if( !numberp(l = params~>l) error( "invalid l" ) )
    dbCreateRect( cellView 1 "drawing"
      list( -2:0 l+2.0:w ) );; diffusion
    dbCreateRect( cellView 2 "drawing"
      list( -1:-1 l+1.0:w+1.0 ) );; implant
    dbCreateRect( cellView 3 "drawing"
      list( 0:-2 l:w+2.0 ) );; poly gate
    return(t)
  )
)
```

3. In CDBA, associate the SKILL procedure with the `nfet` layout cellview:

```
dbCellViewId nfetId;
nfetId=dbOpenCellView( "test" "nfet" "layout" NULL "a" );
if (nfetId != dbcNullCellViewId) {
  dbDefineParamCell( nfetId, "/tmp/nfet.il" );
}
```

This step copies the SKILL file `/tmp/nfet.il` into the database and associates it with the `nfet` layout. The un-encrypted SKILL procedure is no longer needed by the database.

4. In CDBA, create an instance of the `nfet` layout:

```
dbInstId inst;
dbPropValue value;
inst = dbCreateInst( cellViewId, nfetId, NULL );
if (inst != dbcNullCellViewId) {
  val.aFloat = 3.0;
  dbCreateProp( inst, "w", dbcFloatType, val );
  val.aFloat = 1.5;
  dbCreateProp( inst, "l", dbcFloatType, val );
}
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

`nfet` layout and the instance of `nfet` layout appear as

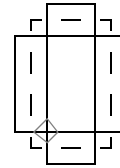
Cellview properties:

`parameters=proplist (w=2.0 l=1.2)`

Variant cellview properties:

`parameters=proplist (w=3.0 l=1.5)`

Encrypted SKILL procedure
is stored in database.



`nfet` layout current

Instance of `nfet` layout current
with properties `w=3` and `l=1.5`

Warnings about Parameterized Cells

Be aware of the following when building and using parameterized cells:

- Defining a parameterized cell does an implicit save of the cellview.
- The database freely creates and deletes `instHeaders` associated with the variants of a parameterized cell.

Abutment Attributes

This section describes the two abutment attributes, `abutName` and `abutRefName`, supported in Parameterized cells.

Abutment Attributes for Parametrized Cells

The `dbPinFigs` owned by a pin can be identified with two names, the `pinFigName` and the `abutName`. Since the `abutName` attribute on each shape has a unique value in the `subMaster` design, you can find a `dbPinFig` based on its `abutName`. In addition, the `dbPinFig` can have an `abutRefName`, which specifies the name of its reference shape. The `abutRefName` attribute need not be unique as multiple shapes can refer to the same reference shape.

Also, the `abutRefName` on a shape must not be identical to its `abutName`, as a shape cannot be its own reference shape.

The functions in this section describe how to set, unset, and get the values of `abutName` and `abutRefName` attributes.

dbSetFigAbutName

BOOLEAN

```
dbSetFigAbutName (
    dbFigId      fig,
    constString  name
);
```

Description

Sets the `abutName` attribute of the `dbPinFig` associated with the given `dbFigId`. If you specify an empty string as the `abutName`, the existing name specified for the `dbPinFig` is cleared.

The following conditions must be met to successfully assign a unique name to the `dbPinFig`:

- The specified `dbFigId` is that of a `dbPinFig`.
- The `dbPinFig` exists in a Pcell.
- The name being assigned is unique within a Pcell submaster.
- The name and reference name assigned to the `dbPinFig` is different.

Arguments

<i>fig</i>	The database identifier of the <code>dbPinFig</code> to which you want to assign a name.
<i>name</i>	The abutment name to be assigned to the <code>dbPinFig</code> .

Return Values

Boolean	Returns <code>TRUE</code> if the name is successfully assigned; <code>FALSE</code> otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetFigAbutRefName

BOOLEAN

```
dbSetFigAbutRefName (
    dbFigId      fig,
    constString  name
);
```

Description

Assigns a reference name to the `dbPinFig` associated with the given `dbFigId`.

The following conditions must be met to successfully assign a unique name to the `dbPinFig`:

- The specified `dbFigId` is that of a `dbPinFig`.
- The `dbPinFig` exists in a Pcell.
- The reference name being assigned to the `dbPinFig` is not the same as the `abutName`.

Arguments

<i>fig</i>	The database identifier of the <code>dbPinFig</code> to which you want to assign a reference name.
<i>name</i>	The reference name to be assigned to the <code>dbPinFig</code> .

Return Values

Boolean	Returns <code>TRUE</code> if the reference name is successfully assigned; <code>FALSE</code> otherwise.

dbGetFigAbutName

```
String  
dbGetFigAbutName (  
    dbFigId      fig  
);
```

Description

Retrieves the name assigned to the `dbPinFig` associated with the given `dbFigId`

The following conditions must be met to successfully retrieve the name of the `dbPinFig`:

- The specified `dbFigId` is that of a `dbPinFig`.
- An `abutName` must have been assigned to the `dbPinFig` with the given `dbFigId`.
- The `dbPinFig` exists in a `Pcell`.

Arguments

<i>fig</i>	The database identifier of the <code>dbPinFig</code> for which you want to retrieve the <code>abutName</code> .

Return Values

<code>String</code>	The <code>abutName</code> of the <code>dbPinFig</code> associated with the given <code>dbFigId</code> . If no <code>abutName</code> is assigned to the <code>dbPinFig</code> associated with the given <code>dbFigId</code> , an empty string is returned.

dbGetFigAbutRefName

String

```
dbGetFigAbutRefName (
    dbFigId      fig
);
```

Description

Retrieves the reference name assigned to the `dbPinFig` associated with the given `dbFigId`

The following conditions must be met to successfully retrieve the reference name of the `dbPinFig`:

- The specified `dbFigId` is that of a `dbPinFig`.
- A reference name must have been assigned to the `dbPinFig` with the given `dbFigId`.
- The `dbPinFig` exists in a Pcell.

Arguments

fig

The database identifier of the `dbPinFig` for which you want to retrieve the reference name.

Return Values

String

The reference name of the `dbPinFig` associated with the given `dbFigId`. If no reference name is assigned to the `dbPinFig` associated with the given `dbFigId`, an empty string is returned.

dbHasFigAbutName

Boolean

```
dbHasFigAbutName (
    dbFigId      fig
);
```

Description

Returns a Boolean value indicating whether a name was assigned to the `dbPinFig` associated with the given `dbFigId`.

The following conditions must be met for this function to return `TRUE`:

- The specified `dbFigId` is that of a `dbPinFig`.
- The `dbPinFig` exists in a `Pcell`.

Arguments

fig

The database identifier of the `dbPinFig` for which you want to confirm if a name was set.

Return Values

Boolean

Returns `TRUE` if a name was set for the `dbPinFig` associated with the given `dbFigId`. Returns `FALSE` if no name was set for the specified `dbPinFig` or if the `dbPinFig` does not exist in a `Pcell`.

dbHasFigAbutRefName

Boolean

```
dbHasFigAbutRefName (  
    dbFigId      fig  
);
```

Description

Returns a Boolean value indicating whether a reference name was assigned to the `dbPinFig` associated with the given `dbFigId`.

The following conditions must be met for this function to return `TRUE`:

- The specified `dbFigId` is that of a `dbPinFig`.
- The `dbPinFig` exists in a `Pcell`.

Arguments

fig

The database identifier of the `dbPinFig` for which you want to confirm if a reference name was set.

Return Values

Boolean

Returns `TRUE` if a reference name was set for the `dbPinFig` associated with the given `dbFigId`. Returns `FALSE` if no reference name was set for the specified `dbPinFig` or if the `dbPinFig` does not exist in a `Pcell`.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindAbutFig

```
dbFigId
dbFindAbutFig(
    dbCellViewId    subMasterId,
    constString      name
);
```

Description

Searches for the `dbPinFig` with the specified name in the given submaster cell and returns its database identifier.

Arguments

<i>subMasterId</i>	The database identifier of the submaster cell in which you want to search for the <code>dbPinFig</code> with the specified name.
<i>name</i>	The name the <code>dbPinFig</code> that you want to search for.

Return Values

<code>dbFigId</code>	The database identifier of the <code>dbPinFig</code> that you are searching for. Returns <code>dbcNullId</code> if the specified name is not that of a <code>dbPinFig</code> or if the <code>dbPinFig</code> does not exist in a Pcell.

Mosaic Functions

A mosaic is a database object that represents an array of instances of one master. A mosaic instance is a database object that represents each master in the array and is associated with a user-defined icon.

The following sections list all CDBA functions available to create, delete, and modify mosaics and mosaic instances, retrieve information about mosaics and mosaic instances, and traverse the hierarchy to generate mosaics.

Mosaic and Mosaic Instance Creation

The following are the CDBA functions available to create mosaics and mosaic instances having various characteristics.

dbCreateSimpleMosaic

```
dbMosaicId  
dbCreateSimpleMosaic(  
    dbCellViewId cellViewId, dbCellViewId masterId, String mosaicName, dbPoint  
    origin, dbOrient orientation, unsigned short rows, unsigned short columns,  
    dbDistance rowSpacing, dbDistance columnSpacing  
);
```

Creates a simple mosaic from the master specified by `masterId` and returns its ID. The name of the instance, specified by the string `mosaicName`, must be unique within the cellview. If you do not want to name the instance, you can pass `NULL` as the `mosaicName` argument. CDBA then names the instance with the unique string representation of a serial number kept by the CDBA for each cellview. The serial number is updated each time an instance is named by CDBA.

The orientation argument specifies the orientation of the instance. Refer to the table associated with `dbCreateInst` for a complete description of the orientation. The argument `origin` specifies the origin of the lower left instance in the array. Rows and columns specify the size of the array. The argument `rowSpacing` specifies the spacing between mosaic element origins in the y direction. The argument `columnSpacing` specifies the space between mosaic element origins in the x direction. If the creation fails, the function returns `dbcNullMosaicId`.

dbCreateDefaultSpacingMosaic

```
dbMosaicId
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbCreateDefaultSpacingMosaic(  
    dbCellViewId cellViewId, dbCellViewId masterId, String mosaicName, dbPoint  
    origin, dbOrient orientation, unsigned short rows, unsigned short columns  
);
```

Creates a simple mosaic, similar to `dbCreateSimpleMosaic`, but row and column spacings are derived from the master cellview. If the master cellview contains a shape on the align layer (`dbcAlignLayer`), it uses the shape's bounding box. Otherwise, it uses the master cellview's bounding box. The row and column spacings are computed from the height and width of the bounding box after it is rotated by the orientation specified.

dbCreateMosaic

```
dbMosaicId  
dbCreateMosaic(  
    dbCellViewId cellViewId, String mosaicName, dbPoint origin, unsigned short  
    rows, unsigned short columns  
);
```

Creates a mosaic of type `dbcEmptyMosaic`, which is a regular array consisting of empty cells. There is no master associated with this mosaic, and its row and column spacing defaults to the uniform spacing of zero. It returns a `mosaicId`. The `rows` and `columns` arguments specify the size of the array. The size of the tile array is equal to the number of rows times the number of columns.

dbCreateMosaicInst

```
dbMosaicInstId  
dbCreateMosaicInst(  
    dbMosaicId mosaicId, dbCellViewId masterId, dbIcon icon  
);
```

Creates a mosaic instance with the specified icon and appends it to the instance list of the mosaic specified by the `mosaicId`.

Mosaic and Mosaic Instance Deletion

The following CDBA functions delete mosaics and mosaic instances.

dbDeleteMosaic

```
Boolean
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbDeleteMosaic(  
    dbMosaicId mosaicId  
);
```

Deletes the mosaic specified by `mosaicId`. Also, it deletes the associated mosaic instance list and tile array. If the mosaic is a parent, then all the descendants are likewise deleted.

See “[CDBA Parent-Child Relationship Functions](#)” on page 979 for more details about parent-child relationships.

dbDeleteMosaicInst

Boolean

```
dbDeleteMosaicInst(  
    dbMosaicInstId mosaicInstId  
);
```

Deletes the mosaic instance specified by `mosaicInstId` from the mosaic instance list. If the tile array contains references to this mosaic instance, they are converted to `dbcEmptyIcon`.

Mosaic Modification

The following CDBA functions modify mosaics.

dbSetRowOffsetByIndex

Boolean

```
dbSetRowOffsetByIndex(  
    dbMosaicId mosaicId, int index, dbDistance offset  
);
```

Sets the row offset of a mosaic by specifying an index that indicates which row you want to set to the specified offset.

dbSetColumnOffsetByIndex

Boolean

```
dbSetColumnOffsetByIndex(  
    dbMosaicId mosaicId, int index, dbDistance offset  
);
```

Sets the column offset of a mosaic by specifying an index that indicates which column you want to set to the specified offset.

dbSetRowOffset

Boolean

```
dbSetRowOffset(  
    dbMosaicId mosaicId, dbDistance offsetArray  
);
```

Sets the row offset of a mosaic. Row offset is the distance between the bottom edge of the mosaic and the top edge of the row. The size of the `offsetArray` must be the same as the number of rows in the array. If the array has uniform spacing and is not a simple array, the row offset reflects the variable spacing with the specified `offsetArray`. The values in the array must increase monotonically.

dbSetColumnOffset

Boolean

```
dbSetColumnOffset(  
    dbMosaicId mosaicId, dbDistance offsetArray  
);
```

Sets the column offset of a mosaic. Column offset is the distance between the left edge of the mosaic and the right edge of the column. The size of the `offsetArray` must equal the number of columns in the mosaic. If the mosaic has uniform spacing and is not a simple array, the column offset reflects the variable spacing with the specified `offsetArray`. The values in the array must increase monotonically.

dbSetUniformRowSpacing

Boolean

```
dbSetUniformRowSpacing(  
    dbMosaicId mosaicId, dbDistance spacing  
);
```

Sets the row spacing of a mosaic to uniform spacing. If the mosaic has variable row spacing, it becomes uniform spacing with the specified `spacing`.

dbSetUniformColumnSpacing

Boolean

```
dbSetUniformColumnSpacing(  
    dbMosaicId mosaicId, dbDistance spacing  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Sets the column spacing of a mosaic to uniform spacing. If the mosaic has variable column spacing, it becomes uniform spacing with the specified `spacing` after this function.

dbSetMosaicName

Boolean

```
dbSetMosaicName(  
    dbMosaicId mosaicId, String mosaicName  
);
```

Changes the name of a mosaic.

dbSetMosaicType

Boolean

```
dbSetMosaicType(  
    dbMosaicId mosaicId, dbMosaicType mosaicType  
);
```

Lets you change the type of a mosaic. Before changing the type, this function checks the tile array of the mosaic to verify whether or not the mosaic can be changed to the type specified. For example, a row-biased array cannot be changed to a regular mosaic if the tile array contains `dbcGoLeftIcon` or `dbcPaddingIcon`.

dbSetMosaicTileArray

Boolean

```
dbSetMosaicTileArray(  
    dbMosaicId mosaicId, dbMosaicTile tileArray, dbMosaicType mosaicType  
);
```

Replaces the tile array with the new tile array if it satisfies the type specified. The size of the tile array must be equal to the number of rows times the number of columns of the mosaic.

dbSetMosaicInstMaster

Boolean

```
dbSetMosaicInstMaster(  
    dbInstId mosaicInstId, dbCellViewId masterId  
);
```

Changes the master of the `mosaicInst` to the one specified by `masterId`.

dbSetMosaicSize

Boolean

```
dbSetMosaicSize(  
    dbMosaicId mosaicId, unsigned int rows, unsigned int columns  
);
```

Changes the number of rows and columns of the mosaic to the specified values. The size of the tile array is decreased or increased to reflect the new size. The tile array is decreased by eliminating the top-most rows or the right-most columns of the array. It is increased by adding rows at the top of the array or columns at the right of the array. The newly added rows or columns contain empty cells.

dbSetMosaicOrigin

Boolean

```
dbSetMosaicOrigin(  
    dbMosaicId mosaicId, dbPoint origin  
);
```

Changes the origin of the array.

Mosaic and Mosaic Instance Information Retrieval

The following CDBA functions retrieve information about mosaics and mosaic instances.

dbGetMosaicType

dbMosaicType

```
dbGetMosaicType(  
    dbMosaicId mosaicId  
);
```

Returns the mosaic type.

dbGetMosaicTileArray

Boolean

```
dbGetMosaicTileArray(  
    dbMosaicId mosaicId, dbMosaicTile pTileArray  
);
```


Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

The tile array of the mosaic is put in `TileArray`, which must be allocated before the call. The size of `TileArray` must match the size of the mosaic.

dbGetMosaicOrigin

```
dbPoint
dbGetMosaicOrigin(
    dbMosaicId mosaicId
);
```

Returns the origin of the mosaic.

dbGetMosaicName

```
String
dbGetMosaicName(
    dbMosaicId mosaicId
);
```

Returns the name of the mosaic as a permanent string.

dbGetMosaicSize

```
Boolean
dbGetMosaicSize(
    dbMosaicId mosaicId, unsigned short pRows, unsigned short pColumns
);
```

Returns the size (number of rows and columns) of the mosaic.

dbGetMosaicNumInst

```
unsigned short
dbGetMosaicNumInst(
    dbMosaicId mosaicId
);
```

Returns the number of rows in the specified mosaic.

dbGetMosaicNumRows

```
unsigned int  
dbGetMosaicNumRows (  
    dbMosaicId    mosaicId  
);
```

Description

Returns the number of rows in the specified mosaic.

Arguments

<i>mosaicId</i>	ID of a mosaic.

Return Value

<i>unsigned int</i>	Number of rows in the specified mosaic.

dbGetMosaicNumColumns

```
unsigned int  
dbGetMosaicNumColumns (  
    dbMosaicId    mosaicId  
);
```

Description

Returns the number of columns in the specified mosaic.

Arguments

<i>mosaicId</i>	ID of a mosaic.

Return Value

<i>unsigned int</i>	Number of columns in the specified mosaic.

dbGetMosaicMaxExt

```
dbDistance  
dbGetMosaicMaxExt(  
    dbMosaicId mosaicId  
);
```

Returns the maximum extension attribute of the mosaic.

dbGetMosaicMasterIcon

```
dbCellViewId  
dbGetMosaicMasterIcon(  
    dbMosaicId mosaicId, dbIcon icon  
);
```

Returns the master cellview of the `mosaicInst`, represented by the given icon.

dbGetMosaicByName

```
dbMosaicId  
dbGetMosaicByName(  
    dbCellViewId cellViewId, String name  
);
```

Returns a mosaic specified by the name in the cellview. If there is no mosaic with that name, it returns NULL.

dbGetMosaicOutline

```
dbBBox *  
dbGetMosaicOutline(  
    dbMosaicId mosaicId  
);
```

Returns the outline bounding box of the mosaic.

dbGetMosaicBBox

```
dbBBox *  
dbGetMosaicBBox(  
    dbMosaicId mosaicId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Returns the bounding box of the mosaic.

dbGetMosaicInstMosaic

```
dbMosaicId  
dbGetMosaicInstMosaic(  
    dbMosaicInstId mosaicInstId  
);
```

Returns the mosaic that `mosaicInstId` belongs to.

dbGetMosaicInstIcon

```
dbIcon  
dbGetMosaicInstIcon(  
    dbMosaicInstId mosaicInstId  
);
```

Returns the icon that represents `mosaicInst` in the master list of the mosaic.

dbGetMosaicInstBBox

```
dbBBox *  
dbGetMosaicInstBBox(  
    dbMosaicInstId mosaicInstId, unsigned short col, unsigned short row  
);
```

Returns the `bBox` of `mosaicInst` located in the `col` column and `row` row. The `col` or `row` specified starts from zero and ends at one less than the number of columns or rows of the mosaic.

dbGetMosaicInstTransform

```
dbTransform *  
dbGetMosaicInstTransform(  
    dbMosaicInstId mosaicInstId, unsigned short col, unsigned short row  
);
```

Returns the transformation of the `mosaicInst` located in the `col` column and `row` row. The `col` or `row` specified starts from zero and ends at one less than the number of columns or rows of the mosaic.

dbGetRowOffsetByIndex

```
dbDistance  
dbGetRowOffsetByIndex(  
    dbMosaicId mosaicId, int index  
);
```

Returns the offset of the row specified by index.

dbGetColumnOffsetByIndex

```
dbDistance  
dbGetColumnOffsetByIndex(  
    dbMosaicId mosaicId, int index  
);
```

Returns the offset of the column specified by index.

dbGetRowOffset

```
dbDistance *  
dbGetRowOffset(  
    dbMosaicId mosaicId  
);
```

Returns the row offset as a `dbDistance` array. The array is allocated in a temporary buffer that is valid only until the next database call.

dbGetColumnOffset

```
dbDistance *  
dbGetColumnOffset(  
    dbMosaicId mosaicId  
);
```

Returns the column offset as a `dbDistance` array. The array is allocated in a temporary buffer that is valid only until the next database call.

dbGetSimpleMosaicSpacing

```
Boolean  
dbGetSimpleMosaicSpacing(  
    dbMosaicId mosaicId, dbDistance pXSpacing, dbDistance pYSpacing  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Returns the column spacing in `XSpacing` and row spacing in `YSpacing`.

dbGetUniformRowSpacing

```
dbDistance  
dbGetUniformRowSpacing(  
    dbMosaicId mosaicId  
);
```

Returns the row spacing if the mosaic has uniform spacing.

dbGetUniformColumnSpacing

```
dbDistance  
dbGetUniformColumnSpacing(  
    dbMosaicId mosaicId  
);
```

Returns the column spacing if the mosaic has uniform spacing.

dbAllocMosaicTileArray

```
dbMosaicTile **  
dbAllocMosaicTileArray(  
    unsigned short rows, unsigned short columns  
);
```

Allocates a tile array of the specified size.

dbFreeMosaicTileArray

```
Boolean  
dbFreeMosaicTileArray(  
    dbMosaicTile tileArray, unsigned short rows, unsigned short columns  
);
```

Frees a tile array allocated by `dbAllocMosaicTileArray`. The size of the specified `tileArray` must match the specified size.

Boolean Predicate Retrieval Functions

The following CDBA functions return values of `TRUE` or `FALSE` in response to queries about mosaics.

dbIsColumnSpacingUniform

Boolean

```
dbIsColumnSpacingUniform(  
    dbMosaicId mosaicId  
);
```

Returns TRUE if the mosaic column spacing is uniform.

dbIsRowSpacingUniform

Boolean

```
dbIsRowSpacingUniform(  
    dbMosaicId mosaicId  
);
```

Returns TRUE if the mosaic row spacing is uniform.

dbIsSimpleMosaic

Boolean

```
dbIsSimpleMosaic(  
    dbMosaicId mosaicId  
);
```

Returns TRUE if the mosaic is a simple mosaic.

dbHasMosaic

Boolean

```
dbHasMosaic(  
    dbCellViewId cellViewId  
);
```

Returns TRUE if there is a mosaic in the cellview.

Traversal and Generator Functions

The following CDBA functions traverse the hierarchy to generate mosaics, instances in a mosaic, and information on elements stored in mosaics.

dbStartGenMosaic

```
dbGenStateId  
dbStartGenMosaic(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all mosaics in a cellview.

dbGenMosaic

```
Boolean  
dbGenMosaic(  
    dbGenStateId pState, dbMosaicId pMosaicId  
);
```

Generates mosaics contained in a cellview.

dbStartGenMosaicToMosaicInst

```
dbGenStateId  
dbStartGenMosaicToMosaicInst(  
    dbMosaicId mosaicId  
);
```

Starts the generator for all instances in a mosaic.

dbGenMosaicToMosaicInst

```
Boolean  
dbGenMosaicToMosaicInst(  
    dbGenStateId pState, dbMosaicInstId pMosaicInstId, dbIcon pIcon  
);
```

Generates instances contained in a mosaic. In a simple mosaic, only one instance is returned. Because this instance is repeated many times over the entire mosaic, it is a `mosaicInst`. Its location, orientation, and bounding box are not meaningful.

The `Icon` part of the name typically indicates the blank character for a simple mosaic. In a complicated mosaic created by the Structure Compiler, `Icon` is the ASCII character used in the tile array file representing this master.

dbStartGenMosaicToElementInfo

```
dbGenStateId  
dbStartGenMosaicToElementInfo(  
    dbMosaicId mosaicId  
);
```

Starts the generator for all element information stored in a mosaic.

dbGenMosaicToElementInfo

```
Boolean  
dbGenMosaicToElementInfo(  
    dbGenStateId pState, dbCellViewId pMasterId, dbIcon pIcon, dbPoint pLocation,  
    dbOrient pOrientation, dbBBox pBBox  
);
```

Generates element information stored in a mosaic. The data returned can create an instance at the same location as the element in the mosaic.

Guides

This section describes functions affecting Guide objects.

About Guides

A guide is a design object. It corresponds to the OpenAccess oaGuide object.

A guide 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.

Routing nets usually replaces guides with wires, or routes. Guides qualify as route objects, so you can add guides to routes.

The following definition applies to guides.

```
typedef cdsBaseIdType      dbGuideId;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsGuided

Boolean

```
dbIsGuided(  
    dbId      guideId  
);
```

Description

Verifies that the *guideId* is a guide object.

Arguments

<i>guideId</i>	The database identifier of the guide object.

Return Values

Boolean	TRUE if the <i>guideId</i> is a guide object. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateGuide

```
dbGuideId
dbCreateGuide (
    dbCellViewId      cvId,
    dbPoint            startPoint,
    dbPoint            endPoint,
    dbLayer            startLayerNum,
    dbLayer            endLayerNum
);
```

Description

Creates a guide in the specified cellview with the specified attributes.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>startPoint</i>	The starting point of the guide.
<i>endPoint</i>	The ending point of the guide. The end point can be the same as the start point.
<i>startLayerNum</i>	The layer on which the guide starts. You can set a starting layer without setting an ending layer. Optional argument. The default layerNum is dbcNullLayer.
<i>endLayerNum</i>	The layer on which the guide ends. You can set an ending layer without setting a starting layer. Optional argument. The default layerNum is dbcNullLayer.

Return Values

dbGuideId	The ID of the new guide.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteGuide

Boolean

```
dbDeleteGuide(  
    dbGuideId      guideId  
);
```

Description

Deletes the *guideId* from the database.

Arguments

<i>guideId</i>	The database identifier of the guide object.

Return Values

Boolean	TRUE if the <i>guideId</i> is deleted. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetGuidePoints

Boolean

```
dbGetGuidePoints (
    dbGuideId      guideId,
    dbPoint         *startPoint,
    dbPoint         *endPoint
);
```

Description

Obtains the start point and the end point of the guide in the specified dbPoint parameters.

Arguments

<i>guideId</i>	The database identifier of the guide object.
<i>*startPoint</i>	Pointer to the starting point of the guide.
<i>*endPoint</i>	Pointer to the ending point of the guide.

Return Values

Boolean	TRUE if the points are returned successfully. FALSE if they are not.

dbSetGuidePoints

Boolean

```
dbSetGuidePoints (
    dbGuideId      guideId,
    dbPoint         startPoint,
    dbPoint         endPoint
);
```

Description

Changes the start and end points of the guide.

Arguments

<i>guideId</i>	The database identifier of the guide object.
<i>startPoint</i>	Starting point of the guide.
<i>endPoint</i>	Ending point of the guide.

Return Values

Boolean	TRUE if the new points are set successfully. FALSE if they are not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetGuideRoute

```
dbRouteId  
dbGetGuideRoute (  
    dbGuideId      guideId  
);
```

Description

Obtains the route to which the guide is attached.

Arguments

<i>guideId</i>	The database identifier of the guide object.

Return Values

<i>dbRouteId</i>	The route to which the guide is attached. If no route has been specified, NULL is returned.

dbIsGuideFollowLayers

Boolean

```
dbIsGuideFollowLayers(  
    dbGuideId      guideId  
);
```

Description

Indicates whether routing must follow the guide layers when attempting to replace this guide with physical routing.

Arguments

<i>guideId</i>	The database identifier of the guide object.

Return Values

Boolean	TRUE if following the guide layers is mandatory. FALSE if following the guide layers is not mandatory.

dbSetGuideFollowLayers

Boolean

```
dbSetGuideFollowLayers (
    dbGuideId      guideId,
    Boolean        followLayer
);
```

Description

Changes the `followLayer` attribute to specify whether routing must follow the guide layers.

Arguments

<i>guideId</i>	The database identifier of the guide object.
<i>followLayer</i>	The attribute specifying whether routing must follow the guide layers.

Return Values

Boolean	TRUE if the setting operation is successful. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenGuide

```
dbGenStateId  
dbStartGenGuide(  
    dbCellViewId    cellViewId  
);
```

Description

Starts guide generation in the cellview. Works in conjunction with dbGenGuide.

Arguments

<i>cellViewId</i>	The database identifier of the cellview.

Return Values

dbGenStateId	The identifier used to track the state of guide generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenGuide

Boolean

```
dbGenGuide (
    dbGenStateId    pState,
    dbGuideId       *pGuideId
);
```

Description

Generates the guides in the cellview. Works in conjunction with dbStartGenGuide.

Arguments

<i>pState</i>	The state of the guide generation.
<i>*pGuideId</i>	Pointer to the guide object identifier in the cellview.

Return Values

Boolean	TRUE if the guide is generated. FALSE if it is not.

pathSegs

This section describes functions affecting pathSeg objects.

About pathSeg objects

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.

```
typedef cdsBaseIdType dbPathSegId;

typedef enum dbEndStyle {
    dbcUnknownEndStyle      = 0,
    dbcTruncateEndStyle     = 1,
    dbcExtendEndStyle       = 2,
    dbcVariableEndStyle     = 3,
    dbcChamferEndStyle      = 4,
    dbcCustomEndStyle       = 5
} dbEndStyle;
```

The dbSegStyle structure corresponds to the OpenAccess oaSegStyle class. The dbSegStyle structure has information about width, beginStyle, endStyle, beginExtension, and endExtension.

```
typedef struct dbSegStyle {
    unsigned int width;
    dbBeginStyle beginStyle;
    dbEndStyle endStyle;
    unsigned int beginExt;
    unsigned int endExt;
    unsigned int beginLeftDiagExt;
    unsigned int beginRightDiagExt;
    unsigned int beginRightHalfWidth;
    unsigned int endLeftDiagExt;
    unsigned int endRightDiagExt;
    unsigned int endRightHalfWidth;
}
```

The following definition applies to dbPathSegRec.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
typedef struct dbPathSegRec {
    dbType          type;
    dbPathSegId     pathSegId;
    size_t          size;
    dbCellViewId    cellViewId;
    dbNetId         net;
    dbPinId         pin;
    dbBBox          bBox;
    dbLayer         layerNum;
    dbPurpose       purposeNum;
    dbPoint         beginPt;
    dbPoint         endPt;
    dbRouteId       route;
    dbSegStyle      *style;
} dbPathSegRec;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsPathSegId

Boolean

```
dbIsPathSegId(  
    dbPathSegId      pathSegId  
);
```

Description

Verifies that the *pathSegID* is a pathSeg object.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.

Return Values

Boolean	TRUE if the <i>pathSegID</i> is a pathSeg object. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreatePathSeg

dbPathSegId

```
dbCreatePathSeg(  
    dbCellViewId      cvId,  
    dbLayer            layerNum,  
    dbPurpose          purposeNum,  
    dbPoint            beginPoint,  
    dbPoint            endPoint,  
    const dbSegStyle   *segStyle  
);
```

Description

Creates a pathSeg in the design with the specified parameters.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>layerNum</i>	The layer number of the pathSeg object.
<i>purposeNum</i>	The purpose number of the pathSeg object.
<i>beginPoint</i>	Beginning point of the segment.
<i>endPoint</i>	Ending point of the segment.
<i>*segStyle</i>	Pointer to the dbSegStyle structure. The structure corresponds to the oaSegStyle class. The parameters for dbSegStyle are the following.

```
typedef struct dbSegStyle{
    unsigned int    width;
    dbEndStyle      beginStyle
    dbEndStyle      endStyle
    unsigned int    beginExt;
    unsigned int    endExt;
    unsigned int    beginLeftDiagExt;
    unsigned int    beginRightDiagExt;
    unsigned int    beginRightHalfwidth;
    unsigned int    endLeftDiagExt;
    unsigned int    endRightDiagExt;
    unsigned int    endRightHalfWidth
}
```

Return Values

dbPathSegId	The identifier of the pathSeg.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeletePathSeg

Boolean

```
dbDeletePathSeg(  
    dbPathSegId      pathSegId  
);
```

Description

Deletes the *pathSegId* from the database.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.

Return Values

Boolean	TRUE if the <i>pathSegId</i> is deleted. FALSE if it is not.

dbConvertPathSegToPolygon

```
dbPolygonId  
dbConvertPathSegToPolygon(  
    dbPathSegId    pathSegId  
);
```

Description

Converts the pathSeg to a polygon. If the conversion fails, the function returns a `NULL` pointer.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.

Return Values

<code>dbPolygonId</code>	The database identifier of the converted polygon.

dbGetPathSegPoints

Boolean

```
dbGetPathSegPoints (
    dbPathSegId      pathSegId,
    dbPoint*          beginPoint,
    dbPoint*          endPoint
);
```

Description

Obtains the beginning and ending points of the segment.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.
<i>beginPoint</i>	The beginning point of the segment.
<i>endPoint</i>	The endpoint of the segment.

Return Values

Boolean	TRUE if the points are obtained successfully. FALSE if they are not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetPathSegBoundary

Boolean

```
dbGetPathSegBoundary(  
    dbPathSegId      pathSegId,  
    unsigned int*    numPoints,  
    dbPoint**        pointArray  
);
```

Description

Obtains the boundary points of the segment.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.
<i>numPoints</i>	The number of points in the segment boundary.
<i>pointArray</i>	The array containing the points of the boundary.

Return Values

Boolean	TRUE if the boundary is obtained successfully. FALSE if it is not.

dbGetPathSegRecBoundary

Boolean

```
dbGetPathSegRecBoundary(  
    dbPathSegRec      *rec,  
    unsigned int*      numPoints,  
    dbPoint**          pointArray  
);
```

Description

Obtains the boundary points of the segment described in the `dbPathSegRec` structure.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

dbPathSegRec

Pointer to the `dbPathSegRec` structure. Following are the data members of `dbPathSegRec`:

```
typedef struct dbPathSegRec {  
    dbType          type;  
    dbPathSegId     pathSegId;  
    size_t          size;  
    dbCellViewId    cellViewId;  
    dbNetId         net;  
    dbPinId         pin;  
    dbBBox          bBox;  
    dbLayer         layerNum;  
    dbPurpose       purposeNum;  
    dbPoint         beginPt;  
    dbPoint         endPt;  
    dbRouteId       route;  
    dbSegStyle      *style;  
} dbPathSegRec;
```

Note: You must specify `beginPt`, `endPt`, and `style`. The rest of the data members are not required by `dbGetPathSegRecBoundary`.

numPoints

The number of points in the segment boundary.

pointArray

The array containing the points of the boundary.

Return Values

Boolean

TRUE if the boundary is obtained successfully.
FALSE if it is not.

dbGetPathSegStyle

Boolean

```
dbGetPathSegStyle(  
    dbPathSegId      pathSegId,  
    dbSegStyle       *style  
);
```

Description

Obtains the segStyle of the pathSeg.

Arguments

pathSegId

The database identifier of the pathSeg object.

*dbSegStyle**

Pointer to the dbSegStyle structure. The structure corresponds to the oaSegStyle class. The parameters of dbSegStyle are the following.

```
typedef struct dbSegStyle{  
    unsigned int    width;  
    dbEndStyle      beginStyle  
    dbEndStyle      endStyle  
    unsigned int    beginExt;  
    unsigned int    endExt;  
    unsigned int    beginLeftDiagExt;  
    unsigned int    beginRightDiagExt;  
    unsigned int    beginRightHalfWidth;  
    unsigned int    endLeftDiagExt;  
    unsigned int    endRightDiagExt;  
    unsigned int    endRightHalfWidth  
}
```


dbGetPathSegRouteTopology

```
dbRouteTopology  
dbGetPathSegRouteTopology(  
    dbPathSegId      pathSegId  
);
```

Description

Obtains the route topology of the segment.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.

Return Values

dbRouteTopology	Returns the enum value of the route topology. The enum values are the following. dbcUnknownRouteTopology (0) dbcNoneRouteTopology (1) dbcStripeRouteTopology (2) dbcStandardCellWireRouteTopology (3) dbcIOWireRouteTopology (4) dbcBlockWireRouteTopology (5) dbcRingRouteTopology (6) dbcPadRingRouteTopology (7) dbcBlockRingRouteTopology (8) dbcCoreWireRouteTopology (9)

dbIsPathSegOrthogonal

Boolean

```
dbIsPathSegOrthogonal(  
    dbPathSegId      pathSegId  
);
```

Description

Determines if the pathSeg is orthogonal.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.

Return Values

Boolean	TRUE if the pathSeg is orthogonal. FALSE if it is not.

dbSetPathSegPoints

Boolean

```
dbSetPathSegPoints (
    dbPathSegId      pathSegId,
    dbPoint           beginPoint,
    dbPoint           endPoint
);
```

Description

Updates the beginning and ending points of the segment.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.
<i>beginPoint</i>	The beginning point of the segment.
<i>endPoint</i>	The ending point of the segment.

Return Values

Boolean	TRUE if the ending points update successfully. FALSE if they do not.

dbSetPathSegStyle

Boolean

```
dbSetPathSegStyle(  
    dbPathSegId      pathSegId,  
    const dbSegStyle *segStyle  
);
```

Description

Sets the segStyle of the segment.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.
<i>*segStyle</i>	Pointer to the dbSegStyle structure. The structure corresponds to the oaSegStyle class.

Return Values

Boolean	TRUE if the segStyle is set successfully. FALSE if it is not.

dbSetPathSegTopology

Boolean

```
dbSetPathSegTopology(  
    dbPathSegId      pathSegId,  
    dbRouteTopology  routeTopology  
);
```

Description

Sets the route topology of the segment to the specified topology.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.
<i>routeTopology</i>	The enum value of the route topology. The enum values are the following. dbcUnknownRouteTopology (0) dbcNoneRouteTopology (1) dbcStripeRouteTopology (2) dbcStandardCellWireRouteTopology (3) dbcIOWireRouteTopology (4) dbcBlockWireRouteTopology (5) dbcRingRouteTopology (6) dbcPadRingRouteTopology (7) dbcBlockRingRouteTopology (8) dbcCoreWireRouteTopology (9)

Return Values

Boolean	TRUE if the topology is set correctly. False if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetPathSegRoute

```
dbRouteId  
dbGetPathSegRoute (  
    dbPathSegId      pathSegId  
);
```

Description

Obtains the route of the path segment.

Arguments

<i>pathSegId</i>	The database identifier of the pathSeg object.

Return Values

dbRouteId	The routeID of the pathSeg.

Vias

This section describes functions affecting Via objects.

About Vias

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.

There are two types of vias, standard vias and custom vias. A standard via has a predefined set of via parameters. You can tailor the parameters to create a variety of vias that apply to a variety of connections. A standard via references a standard via definition.

A custom via is similar to an instance in that the custom via definition references another design. Custom vias often include a set of parameter values, so you can tailor the master via for each specific via.

Commonly, a via is associated with routes, although a via can represent a pin figure.

Vias always have a viaHeader which contains the attributes that are common to all vias of a specific via definition.

The following definitions apply to vias.

```
typedef cdsBaseIdType dbViaId;

typedef enum dbRouteStatus {
    dbcUnknownRouteStatus      = 0,
    dbcNormalRouteStatus       = 1,
    dbcFixedRouteStatus        = 2,
    dbcLockedRouteStatus       = 3
} dbRouteStatus;

typedef enum dbRouteTopology {
    dbcUnknownRouteTopology      = 0,
    dbcNoneRouteTopology        = 1,
    dbcStripeRouteTopology      = 2,
    dbcStandardCellWireRouteTopology = 3,
    dbcIOWireRouteTopology      = 4,
    dbcBlockWireRouteTopology    = 5,
    dbcRingRouteTopology        = 6,
    dbcPadRingRouteTopology     = 7,
    dbcBlockRingRouteTopology    = 8,
    dbcCoreWireRouteTopology    = 9
}
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
} dbRouteTopology;

typedef enum dbViaDirection {
    dbcLayer1ToLayer2Direction    = 0,
    dbcLayer2ToLayer1Direction    = 1
} dbViaDirection;
```

The following definition applies to dbViaRec.

```
typedef struct dbViaRec {
    dbType      type;
    dbViaId     viaId;
    dbBBox      bBox;
    dbCellViewId cellViewId;
} dbViaRec;
```


Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsViald

Boolean

```
dbIsViaId(  
    dbId      viaId  
);
```

Description

Verifies that the *viaId* is a via object. The via object can be either a standard or custom via.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

Boolean	TRUE if the <i>viaId</i> is a via object. FALSE if it is not a via object.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteVia

Boolean

```
dbDeleteVia(  
    dbViaId    viaId  
);
```

Description

Deletes the *viaId* from the database.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

Boolean	TRUE if the <i>viaId</i> is deleted. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetViaViaHeader

```
dbViaHeaderId  
dbGetViaViaHeader(  
    dbViaId      viaId  
);
```

Description

Returns the viaHeader associated with the via. The viaHeader includes the attributes that are common to all vias of the particular via definition.

For vias of via definitions that have parameters, the function returns the via subheader.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

dbViaHeaderId	The viaHeader associated with the via.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetViaViaDef

```
techViaDefId  
dbGetViaViaDef(  
    dbViaId    viaId  
);
```

Description

Returns the via definition associated with the `viaId`.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

<code>techViaDefId</code>	The via definition associated with the via.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetViaViaDefName

String

```
dbGetViaViaDefName(  
    dbViaId    viaId  
);
```

Description

Returns the name of the via definition associated with the `viaId`.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

String	The name of the via definition associated with the via. Returns a null string if unable to obtain the name of the via definition.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetViaRoute

```
dbRouteId
dbGetViaRoute (
    dbViaId    viaId
);
```

Description

Returns the route to which the `viaId` belongs, if any.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

<code>dbRouteId</code>	The route to which the via belongs.

dbGetViaRouteStatus

```
dbRouteStatus  
dbGetViaRouteStatus(  
    dbViaId    viaId  
);
```

Description

Returns the current route status of the `viaId`.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

<code>dbRouteStatus</code>	Returns the enum value of the route status. The enum values are the following. <code>dbcUnknownRouteStatus</code> <code>dbcNormalRouteStatus</code> <code>dbcFixedRouteStatus</code> <code>dbcLockedRouteStatus</code>

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetViaDirection

```
dbViaDirection  
dbGetViaDirection(  
    dbViaId      viaId  
);
```

Description

Returns the routing direction of the `viaId`.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

<code>dbViaDirection</code>	The routing direction of the via object. The values are the following. <code>dbcUnknownViaDirection</code> <code>dbcLayer1ToLayer2Direction</code> <code>dbcLayer2ToLayer1Direction</code>

dbSetViaRouteStatus

Boolean

```
dbSetViaRouteStatus (
    dbViaId      viaId,
    dbRouteStatus routeStatus
);
```

Description

Sets the routing status of the `viaId`.

Arguments

<i>viaId</i>	The database identifier of the via object.
<i>routeStatus</i>	The enum value of the route status. The enum values are the following. dbcUnknownRouteStatus (0) dbcNormalRouteStatus (1) dbcFixedRouteStatus (2) dbcLockedRouteStatus (3)

Return Values

Boolean	TRUE if the routing status is set successfully. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetViaDirection

Boolean

```
dbSetViaDirection(  
    dbViaId      viaId,  
    dbViaDirection viaDirection  
);
```

Description

Sets the routing direction of the `viaId`.

Arguments

<i>viaId</i>	The database identifier of the via object.
<i>viaDirection</i>	The routing direction of the via object. The values are the following. dbcUnknownViaDirection dbcLayer1ToLayer2Direction dbcLayer2ToLayer1Direction

Return Values

Boolean	TRUE if the routing direction is set successfully. FALSE if it is not.

dbStartGenViaHeaderToVia

```
dbGenStateId  
dbStartGenViaHeaderToVia(  
    dbViaHeaderId    viaHeaderId  
);
```

Description

Starts via generation for the specified via header. Works in conjunction with function dbGenViaHeaderToVia.

Arguments

<i>viaHeaderId</i>	The database identifier of the via header.

Return Values

dbGenStateId	The identifier used to track the state of via generation for the specified header.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenViaHeaderToVia

Boolean

```
dbGenViaHeaderToHeader(  
    dbGenStateId  pState,  
    dbViaId       *pviaId  
);
```

Description

Generates the vias. Works in conjunction with function dbStartGenViaHeaderToVia.

Arguments

<i>pState</i>	The identifier used to track the state of via generation.
<i>*pviaId</i>	The <code>viaId</code> generated by the function.

Return Values

Boolean	TRUE if the vias successfully generate. FALSE if it does not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsCustomViaId

Boolean

```
dbIsCustomViaId(  
    dbId      viaId  
);
```

Description

Verifies that the given *viaId* is a custom via object.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

Boolean	TRUE if the <i>viaId</i> refers to a custom via object. FALSE if it is not a custom via object.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateCustomVia

```
dbViaId
dbCreateCustomVia (
    dbCellViewId      cvId,
    techViaDefId      viaDefId,
    dbTransform        transform,
    unsigned int       numParams,
    const dbParamRec*  params
);
```

Description

Creates a new custom via.

Arguments

<i>cvId</i>	The database cellview identifier of the custom via object.
<i>viaDefId</i>	The identifier of the viaDef associated with the via.
<i>transform</i>	The transform to use to create the via.
<i>numParams</i>	The number of custom via parameters.
<i>params</i>	A pointer to the populated parameter array.

Return Values

<i>dbViaId</i>	The database ID of the new custom via.

dbGetCustomViaParams

Boolean

```
dbGetCustomViaParams (
    dbViaId      viaId
    unsigned int* numParams,
    dbParamRec** params
);
```

Description

Returns the custom via parameters for the custom via.

Arguments

<i>viaId</i>	The database identifier of the via object.
<i>numParams</i>	The number of custom via parameters.
<i>params</i>	A pointer to the populated custom parameter array.

Return Values

Boolean	TRUE if the custom via parameters are successfully obtained. FALSE if they are not.

dbSetCustomViaParams

Boolean

```
dbSetCustomViaParams (
    dbViaId          viaId
    unsigned int      numParams,
    const dbParamRec* params
);
```

Description

Sets the custom via parameters for the custom via.

Arguments

<i>viaId</i>	The database identifier of the via object.
<i>numParams</i>	The number of custom via parameters.
<i>params</i>	A pointer to the populated custom parameter array.

Return Values

Boolean	TRUE if the parameters are set successfully. FALSE if they are not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsStdViald

Boolean

```
dbIsStdViaId(  
    dbId      viaId  
);
```

Description

Verifies that the `viaId` is a standard via object.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

Boolean	TRUE if the <i>viaId</i> refers to a standard via object. FALSE if it is not a standard via object.

dbCreateStdVia

```
dbViaId  
dbCreateStdVia(  
    dbCellViewId    cvId,  
    techViaDefId    viaDefId,  
    dbTransform      transform,  
    unsigned int     numParams,  
    const dbViaParam *viaParam  
);
```

Description

Creates a new standard via.

Arguments

<i>cvId</i>	The database cellview identifier of the custom via object.
<i>viaDefId</i>	The identifier of the via definition associated with the via.
<i>transform</i>	The transform to use to create the via.
<i>numParams</i>	The number of custom via parameters.
<i>*viaParam</i>	A pointer to the populated parameter array.

Return Values

<i>dbViaId</i>	The database identifier of the new standard via.

Example

The following are examples of creating a standard via and setting parameters of a standard via.

```
Boolean  
setStdViaParams(dbViaId viaId)  
{  
    const dbViaParam *viaParams = dbGetStdViaViaParam(viaId);  
    dbStdViaParams* stdViaParams  
        = dbGetViaParamStdViaParams((dbViaParam*)viaParams);  
  
    //Set via parameter with dbUnit. Not userUnit.  
    stdViaParams->layer1Enc.x = 60;  
    stdViaParams->layer1Enc.y = 6;  
    dbSetStdViaParamDefault(viaParams, dbcLayer1EncViaParamType, FALSE);  
}
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
stdViaParams->cutWidth = stdViaParams->cutHeight = 310;
dbSetStdViaParamDefault(viaParams, dbcCutWidthViaParamType, FALSE);
dbSetStdViaParamDefault(viaParams, dbcCutHeightViaParamType, FALSE);

stdViaParams->cutRows = stdViaParams->cutColumns = 2;
dbSetStdViaParamDefault(viaParams, dbcCutRowsViaParamType, FALSE);
dbSetStdViaParamDefault(viaParams, dbcCutColumnsViaParamType, FALSE);

if (!dbSetStdViaParam(viaId, (dbViaParam *)viaParams))
    return error("setStdViaParams: Failed to set via params to a stdVia.\n");
return TRUE;
}

Boolean
createVia(void)
{
    dbCellViewId cellView;
    const oaString testLibName = getLibName();

    cellview = dbOpenCellViewByType(testLibName, "Atop", "layout",
                                     "maskLayout", "w", NULL);

    techFileId tfId = techGetCellViewTechFile(topRep);

    techViaDefId viaDef1 = techFindViaDefByName(tfId, "M2xM1xG");

    dbViaId via1_id = dbCreateStdVia(cellview, viaDef1, dbvIdentTransform, NULL);

    // Creates a stdVia with a specified viaParam.
    dbViaParam *viaParam = techGetStdViaDefParams(viaDef1);

    // Sets cutWidth to its default value 0.13 but turns its defFlag bit on.
    // Supposedly dbCreateStdVia() should create a stdVia with setDefault for
    // cutWidth.
    dbStdViaParams *stdViaParam = dbGetViaParamStdViaParams(viaParam);
    stdViaParam->cutWidth = 130;

    dbSetStdViaParamDefault(viaParam, dbcCutWidthViaParamType, FALSE);
    stdViaParam->cutRows = 2;
    dbSetStdViaParamDefault(viaParam, dbcCutRowsViaParamType, FALSE);
    dbViaId via2_id = dbCreateStdVia(cellview, viaDef1, dbvIdentTransform,
    viaParam);

    const dbViaParam *viaParam_2 = dbGetStdViaViaParam(via2_id);
    stdViaParam = dbGetViaParamStdViaParams((dbViaParam *) viaParam_2);

    if (!dbHasStdViaParamDefault(viaParam_2, dbcCutColumnsViaParamType))
        return error("cutColumns should be setDefault for via2_id.\n");

    if (!dbHasStdViaParamDefault(viaParam_2, dbcCutWidthViaParamType))
        return error("cutWidth should setDefault for via2_id.\n");
    if (stdViaParam->cutWidth != 130)
        return error("The value of cutWidth of stdVia_2 is not 0.13(default).\n");

    // To modify a placed stdVia with a non-default parameter value.
    if (!setStdViaParams(via1_id))
        return error("Failed to set partameters for stdVia (M4xM3xG).\n");
}
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbSave (cellview) ;  
dbClose (cellview) ;  
  
return TRUE;
```

dbGetStdViaViaParam

```
dbViaParam*  
dbGetStdViaViaParam (  
    dbViaId      viaId*  
);
```

Description

Returns the standard via parameters for the standard via.

Arguments

<i>viaId</i>	The database identifier of the via object.

Return Values

<i>dbViaParam*</i>	A pointer to the standard via parameter array.

dbSetStdViaParam

Boolean

```
dbSetStdViaParam
    dbViaId          viaId<
    const dbViaParam *viaParam
);
```

Description

Sets the standard via parameters for the standard via.

Arguments

<i>viaId</i>	The database identifier of the via object.
<i>numParams</i>	The number of custom via parameters.
<i>*viaParam</i>	A pointer to the populated standard parameter array.

Return Values

Boolean	TRUE if the parameters are set successfully. FALSE if they are not.

Example

See [dbCreateStdVia](#) on page 642.

Routes

This section describes functions affecting Route objects.

About Routes

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.

The following definition applies to routes.

```
typedef cdsBaseIdType      dbRouteId;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbIsRouteId

Boolean

```
dbIsRouteId(  
    dbId      objId  
);
```

Description

Verifies that the *objId* is a route object.

Arguments

<i>objId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if the <i>objId</i> is a route object. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateRoute

```
dbRouteId
dbCreateRoute (
    dbCellViewId      cvId,
    dbNetId            netId,
    Boolean            isGlobal,
    dbRouteStatus      status
);
```

Description

Creates a route in the specified cellview with the specified attributes.

The function throws an exception if you specify a net from another cellview.

Arguments

<i>cvId</i>	The database identifier of the cellview.
<i>netId</i>	The database identifier of the net.
<i>isGlobal</i>	TRUE if the route is a global route. FALSE if it is not.
<i>dbRouteStatus</i>	The enum value of the route status. The enum values are the following. dbcUnknownRouteStatus (0) dbcNormalRouteStatus (1) dbcFixedRouteStatus (2) dbcLockedRouteStatus (3)

Return Values

<i>dbRouteId</i>	The identifier of the new route.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteRoute

Boolean

```
dbDeleteRoute(  
    dbRouteId      routeId  
);
```

Description

Deletes the *routeId* from the database.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if the <i>routeId</i> is deleted. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetRouteObjects

Boolean

```
dbGetRouteObjects (
    dbRouteId      routeId,
    unsigned int*   numObjects,
    dbId**          routeObjects
);
```

Description

Returns the objects of the route in a `dbId` array.

Arguments

<i>routeId</i>	The database identifier of the route object.
<i>numObjects</i>	The number of objects in the array.
<i>routeObjects</i>	The array containing the objects of the route.

Return Values

Boolean	TRUE if the route objects are returned successfully. FALSE if they are not.

dbGetRouteNumObjects

```
unsigned int  
dbGetRouteNumObjects(  
    dbRouteId    routeId  
);
```

Description

Obtains the number of routing objects in the route. Routing objects can include guide, pathSeg, customVia, and stdVia objects.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

unsigned int	The number of objects in the route object.

dbGetRouteStatus

```
dbRouteStatus  
dbGetRouteStatus(  
    dbRouteId      routeId  
);
```

Description

Obtains the route status of the route.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

<i>dbRouteStatus</i>	The enum value of the route status. The enum values are the following. dbcUnknownRouteStatus (0) dbcNormalRouteStatus (1) dbcFixedRouteStatus (2) dbcLockedRouteStatus (3)

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetRouteStartConn

```
dbId  
dbGetRouteStartConn(  
    dbRouteId    routeId  
);
```

Description

Obtains the identifier of the connection object at which the route starts.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

<i>dbId</i>	The database identifier of the object at which the route starts. The identifier is <code>dbcNullId</code> if no connection object exists.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetRouteEndConn

```
dbId  
dbGetRouteEndConn(  
    dbRouteId    routeId  
);
```

Description

Obtains the identifier of the connection object at which the route ends.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

<i>dbId</i>	The database identifier of the object at which the route ends. The identifier is <code>dbcNullId</code> if no connection object exists.

dbIsRouteGlobal

Boolean

```
dbIsRouteGlobal(  
    dbRouteId      routeId  
);
```

Description

Verifies whether the route is a global route.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if the route is marked as a global route. FALSE if it is not.

dbIsRouteHasGuide

Boolean

```
dbIsRouteHasGuide (
    dbRouteId      routeId
);
```

Description

Verifies whether the route contains one or more guides.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if the route contains one or more guides. FALSE if it does not.

dbIsRouteContiguous

Boolean

```
dbIsRouteContiguous (
    dbRouteId      routeId
);
```

Description

Verifies whether all objects in the route are contiguous. Route objects are pathSegs, vias, and guides.

A route can span multiple layers. So, the function determines contiguity based on the point at which each route object connects to the adjacent route object and the layer on which each connects. A route is contiguous if the end point of each object matches the beginning point of the adjacent object and the two points are on the same layer.

For vias, the via origin is both the beginning point and the ending point. The via direction determines the layer. For example, a via direction is set to `oacLayer1ToLayer2ViaDirection`. Layer1 must match the object before the via. Layer2 must match the object after the via.

For guides with no specified starting or ending layer, the function assumes the start/end layer matches any layer specified for the adjacent object.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if all objects in the route are contiguous. FALSE if they are not.
	The function returns TRUE for single-object routes and returns FALSE for zero-object or empty routes.

dbSetRouteStatus

Boolean

```
dbSetRouteStatus (
    dbRouteId      routeId,
    dbRouteStatus  routeStatus
);
```

Description

Sets the route status for the specified route.

Arguments

<i>routeId</i>	The database identifier of the route object.
<i>dbRouteStatus</i>	The enum value of the route status. The enum values are the following. dbcUnknownRouteStatus (0) dbcNormalRouteStatus (1) dbcFixedRouteStatus (2) dbcLockedRouteStatus (3)

Return Values

Boolean	TRUE if the route status is set successfully. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetRouteObjects

Boolean

```
dbSetRouteObjects (
    dbRouteId      routeId,
    unsigned int    numObjects,
    dbId*           routeObjects
);
```

Description

Updates the route by replacing the existing route objects with new route objects specified in the routeObjects array.

Arguments

<i>routeId</i>	The database identifier of the route object.
<i>numObjects</i>	The number of objects in the array.
<i>routeObjects</i>	The array of new route objects.

Return Values

Boolean	TRUE if the new route objects are passed in successfully. FALSE if they are not.

dbSetRouteGlobal

Boolean

```
dbSetRouteGlobal(  
    dbRouteId      routeId,  
    Boolean         global  
);
```

Description

Sets or resets the `global` attribute of the route.

Arguments

<i>routeId</i>	The database identifier of the route object.
Boolean	The <code>global</code> attribute value to set for the route.

Return Values

Boolean	TRUE if the <code>global</code> attribute is set successfully. FALSE if it is not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetRouteStartConn

Boolean

```
dbSetRouteStartConn(  
    dbRouteId    routeId,  
    dbId         objectId  
);
```

Description

Connects the specified object to the starting point of the specified route. If the route already has a starting connection object, the route is disconnected first then connected to the specified object.

Arguments

<i>routeId</i>	The database identifier of the route object.
<i>objectId</i>	The database identifier of the object to connect.

Return Values

Boolean	TRUE if the object connects successfully. FALSE if it does not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetRouteEndConn

Boolean

```
dbSetRouteEndConn (
    dbRouteId      routeId,
    dbId           objectId
);
```

Description

Connects the specified object to the ending point of the specified route. If the route already has an ending connection object, the route is disconnected first then connected to the specified object.

Arguments

<i>routeId</i>	The database identifier of the route object.
<i>objectId</i>	The database identifier of the object to connect.

Return Values

Boolean	TRUE if the object connects successfully. FALSE if it does not.

dbUnsetRouteStartConn

Boolean

```
dbUnsetRouteStartConn(  
    dbRouteId      routeId  
);
```

Description

Disconnects the route from its starting connection object.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if the route disconnects successfully. FALSE if it does not.

dbUnsetRouteEndConn

Boolean

```
dbUnsetRouteEndConn(  
    dbRouteId      routeId  
);
```

Description

Disconnects the route from its ending connection object.

Arguments

<i>routeId</i>	The database identifier of the route object.

Return Values

Boolean	TRUE if the route disconnects successfully. FALSE if it does not.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetNumRoutes

```
unsigned int* numRoutes  
dbGetNumRoutes(  
    dbCellViewId      cvId,  
    );
```

Description

Obtains the number of routes in the cellview.

Arguments

<i>cvId</i>	The database identifier of the cellview.

Return Values

<i>numRoutes</i>	The number of routes in the cellview.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenRoute

```
dbGenStateId  
dbStartGenRoute(  
    dbCellViewId    cvId  
);
```

Description

Starts generating all the routes in the cellview. Works in conjunction with function dbGenRoute.

Arguments

<i>cvId</i>	The database identifier of the cellview.

Return Values

<i>dbGenStateId</i>	The identifier used to track the state of route generation in the cellview.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenRoute

Boolean

```
dbGenRoute (
    dbGenStateId    pState,
    dbRouteId*      routeId
);
```

Description

Generates all the routes in the cellview. Works in conjunction with function dbStartGenRoute.

Arguments

<i>pState</i>	The identifier used to track the state of route generation in the cellview.
<i>routeId</i>	The identifier of each newly generated route.

Return Values

Boolean	TRUE if the routes generate successfully. FALSE if they do not.

dbStartGenNetToRoute

```
dbGenStateId  
dbStartGenNetToRoute(  
    dbNetId      netId  
);
```

Description

Starts generating all the routes for the net. Works in conjunction with function dbGenNetToRoute.

Arguments

<i>netId</i>	The database identifier of the net.

Return Values

dbGenStateId	The identifier used to track the state of route generation for the net.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenNetToRoute

Boolean

```
dbGenNetToRoute (  
    dbGenStateId      pState,  
    dbRouteId*        routeId  
);
```

Description

Generates all the routes for the net. Works in conjunction with function dbStartGenNetToRoute.

Arguments

<i>pState</i>	The identifier used to track the state of route generation for the net.
<i>routeId</i>	The identifier of each newly generated route.

Return Values

Boolean	TRUE if the routes generate successfully. FALSE if they do not.

dbStartGenShapeToConnRoute

```
dbGenStateId  
dbStartGenShapeToConnRoute (  
    dbShapeId      shapeId  
);
```

Description

Starts generating all the routes that connect to the shape. Works in conjunction with function dbGenShapeToConnRoute.

Arguments

<i>shapeId</i>	The database identifier of the shape.

Return Values

dbGenStateId	The identifier used to track the state of route generation for the shape.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenShapeToConnRoute

Boolean

```
dbGenShapeToConnRoute (  
    dbGenStateId      pState,  
    dbRouteId*        routeId  
);
```

Description

Generates all the routes that connect to the shape. Works in conjunction with function dbStartGenShapeToConnRoute.

Arguments

<i>pState</i>	The identifier used to track the state of route generation for the shape.
<i>routeId</i>	The identifier of each newly generated route.

Return Values

Boolean	TRUE if the routes generate successfully. FALSE if they do not.

dbStartGenPinToConnRoute

```
dbGenStateId  
dbStartGenPinToConnRoute(  
    dbPinId    pinId  
);
```

Description

Starts generating all the routes that connect to the pin. Works in conjunction with function dbGenPinToConnRoute.

Arguments

<i>pinId</i>	The database identifier of the pin.

Return Values

dbGenStateId	The identifier used to track the state of route generation for the pin.

dbGenPinToConnRoute

Boolean

```
dbGenPinToConnRoute (  
    dbGenStateId    pState,  
    dbRouteId*      routeId  
);
```

Description

Generates all the routes that connect to the pin. Works in conjunction with function dbStartGenPinToConnRoute.

Arguments

<i>pState</i>	The identifier used to track the state of route generation for the pin.
<i>routeId</i>	The identifier of each newly generated route.

Return Values

Boolean	TRUE if the routes generate successfully. FALSE if they do not.

dbStartGenTermToConnRoute

```
dbGenStateId  
dbStartGenTermToConnRoute(  
    dbTermId      termId  
);
```

Description

Starts generating all the routes that connect to the term. Works in conjunction with function dbGenTermToConnRoute.

Arguments

<i>termId</i>	The database identifier of the term.

Return Values

dbGenStateId	The identifier used to track the state of route generation for the term.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenTermToConnRoute

Boolean

```
dbGenTermToConnRoute (  
    dbGenStateId      pState,  
    dbRouteId*        routeId  
);
```

Description

Generates all the routes that connect to the term. Works in conjunction with function dbStartGenTermToConnRoute.

Arguments

<i>pState</i>	The identifier used to track the state of route generation for the term.
<i>routeId</i>	The identifier of each newly generated route.

Return Values

Boolean	TRUE if the routes generate successfully. FALSE if they do not.

dbStartGenInstTermToConnRoute

```
dbGenStateId  
dbStartGenInstTermToConnRoute (  
    dbInstTermId      instTermId  
);
```

Description

Starts generating all the routes that connect to the instTerm. Works in conjunction with function dbGenInstTermToConnRoute.

Arguments

<i>instTermId</i>	The database identifier of the instTerm.

Return Values

dbGenStateId	The identifier used to track the state of route generation for the instTerm.

dbGenInstTermToConnRoute

Boolean

```
dbGenInstTermToConnRoute (  
    dbGenStateId      pState,  
    dbRouteId*        routeId  
);
```

Description

Generates all the routes that connect to the instTerm. Works in conjunction with function dbStartGenInstTermToConnRoute.

Arguments

<i>pState</i>	The identifier used to track the state of route generation for the instTerm.
<i>routeId</i>	The identifier of each newly generated route.

Return Values

Boolean	TRUE if the routes generate successfully. FALSE if they do not.

CDBA Connectivity Functions

Connectivity is the logical and physical connection of signals in one cellview to signals in other cellviews. This chapter defines objects, relationships, and attributes that implement logical and physical connectivity and that are accessible through the programming interface. In addition, it describes the rules of use and limitations imposed by the current implementation.

The following sections further define the concepts involved in connectivity and describe the relationships between these concepts in more detail.

- [Object Classes and Their Members](#) on page 680
- [Figures](#) on page 681
- [Connections in Cellviews](#) on page 681
- [Any Instance](#) on page 681
- [Nets](#) on page 684
- [Signals](#) on page 705
- [Terminals](#) on page 711
- [Instance Terminals](#) on page 740
- [Pins](#) on page 748
- [BusDef and BitOrder](#) on page 766
- [Cellview Information](#) on page 798
- [Equivalent Connectivity Time Functions](#) on page 803
- [Net and Terminal Name Syntax](#) on page 808
- [Name and Vector Expression Expansion](#) on page 810
- [Automatic Database Deletions](#) on page 811
- [Constraints on Database Objects](#) on page 811
- [Restrictions to the Interface](#) on page 812
- [Connectivity Example Function](#) on page 812

Object Classes and Their Members

An object class is a data type abstraction that groups distinct but related objects into a class (an array) according to the attributes or common generators that the objects share. Example object classes are geometric shapes, instances, and figures.

The following table shows the enumerated constants and ID types associated with each object class used in connectivity.

Object Classes Used in Connectivity

Name	Enumerated constant	ID type
Signal	dbcSigType	dbSigId
Net	dbcNetType	dbNetId
Any Instance	–	dbAnyInstId
Instance	dbcInstType	dbInstId
Mosaic Instance	dbcMosaicInstType	dbMosaicInstId
Figure	–	dbFigId
Shape	–	dbShapeId
Mosaic	dbcMosaicType	dbMosaicId
Terminal	dbcTermType	dbTermId
Instance Terminal	dbcInstTermType	dbInstTermId
Pin	dbcPinType	dbPinId

A member is a specific object in an array of objects. Examples of object members include member nets, member instances, member connections (connected instance terminals), and member terminals.

When used in a function name, the `mem` part implies the presence of an ID and an integer index that specifies which member of the object array (counting from the left-most member) is under consideration.

For each member, the index is zero-based, starting with the left-most member when the list of objects is expanded. The left-most bit is commonly used as the most significant bit when the data on a bus is treated as a number.

Figures

A figure is a class of database objects, including instances and geometric shapes, that are accessible from nets or pins. Figures are the geometries or instances that implement a net or pin. A net without figures is invisible when viewed with the graphics editor. A pin cannot exist without a figure. A figure is represented in a name by the `fig` part.

Connections in Cellviews

A terminal on a cellview represents a net containing a single electrical signal or a group of electrical signals. Terminals and pins indicate which nets in a cellview are available for connection when an instance of a cellview is placed.

An instance terminal is an instance of a master terminal in a given cellview. When you place an instance of a cellview, you can connect a net to its corresponding instance terminal. The associated signals in this connection between the two cellviews start from the left-most bit.

Instance terminals and nets implement logical connections. Instance pins and nets implement physical connections.

A connected instance terminal is the logical connection between an instance, an instance terminal, and a net within a cellview, and between the two nets in the master and instance cellviews. The net, instance, and instance terminal objects are in the instance cellview, and the master terminal and its net are in the master cellview of the instance.

A connected instance pin is the physical connection between a net, an instance terminal, and a pin. The connection occurs where the instance is placed. The net connected to an instance terminal and the net connected to the associated instance pin must contain the same electrical signals.

Any Instance

The term any instance refers to a class of database objects that includes instances and mosaic instances. Any instance is represented in a name by the `anyInst` part of the name. When any type of instance appears, `anyInst` is used in the appropriate place. All types of instances must have unique names within any specified cellview.

A member instance refers to a specific member of an instance. When used in a function name, `memInst` implies the presence of an `anyInstId` and an integer index. The index specifies which member of the instance (starting from zero) is under consideration. A member instance is represented in a name by the `memInst` part of the name.

Instances

An instance is a database object that implies there is another cellview inside the cellview containing the instance. An instance has a particular location and orientation. The instance has a name, a type, and a list of connections. Each instance name in the cellview must be unique.

There is, at most, one connection on an instance for each terminal in the master cellview of the instance. In addition, instances are user-definable and have a purpose associated with them.

The integer attribute of an instance, called `numInst`, is the number of times the master of the instance is logically instantiated. When `numInst` is greater than one, the instance is essentially a vector of instances represented with one object. This is also called an iterated instance. Each connection of an iterated instance connects all bits for each terminal of all the implied instances of the iterated instance. There is, at most, one instance connected to a net or pin; nets and pins cannot support a vector of instances.

Functions Acting on Instances

The following functions act on instances.

dbGetMemInstName

```
String
dbGetMemInstName (
    dbAnyInstId anyInstId, int instIndex
);
```

Returns the name of the member instance.

dbStartGenAnyInstToConn

```
dbGenStateId
dbStartGenAnyInstToConn (
    dbAnyInstId anyInstId
);
```

Starts the generator for connected instance terminals attached to this instance, if any. One instance terminal on an iterated instance represents all instance terminals for the implied instances of the iterated instance.

dbGenAnyInstToConn

Boolean

```
dbGenAnyInstToConn(  
    dbGenStateId state, dbInstTermId pInstTermId  
);
```

Generates connected instance terminals attached to this instance, if any. One instance terminal on an iterated instance represents all instance terminals for the implied instances of the iterated instance.

dbStartGenAnyInstToInstTerm

dbGenStateId

```
dbStartGenAnyInstToInstTerm(  
    dbAnyInstId anyInstId  
);
```

Starts the generator for instance terminals (both connected and unconnected) for this instance, if any.

dbGenAnyInstToInstTerm

Boolean

```
dbGenAnyInstToInstTerm(  
    dbGenStateId state, dbInstTermId pInstTermId  
);
```

Generates instance terminals (both connected and unconnected) for this instance, if any.

The following six functions for any instance are documented in [“Instance and Hierarchy Functions”](#) on page 463.

- dbGetAnyInstName
- dbSetAnyInstName
- dbGetAnyInstBaseName
- dbSetAnyInstBaseName
- dbGetAnyInstNumInst
- dbSetAnyInstNumInst

Nets

A net is a database object representing the physical implementation of a single electrical signal or group of electrical signals. A net implements the connections to instance pins and to pins of the cellview containing the net. A net can have pins, terminals, instance terminals, internal geometries that implement connections, and instance pins associated with it.

Nets and Signals

A net has a name and a number of bits. When you explicitly list the names of the group members, the net name is formed that implements a group of signals. For example, a net with the name A, B, C contains three signals named A, B, and C, respectively. The same member can occur more than once. For example, a net A, B, A contains two signals named A and a signal named B.

A signal can also occur in more than one net; a signal can be physically implemented (or wired) by more than one net. For example, net A, B and net C, A are both implementing signal A. This signal is referred to as member index (or bit) zero and index one of nets A, B and C, A, respectively. Consequently, the term member net, represented in a name by the `memNet` part, is introduced to refer to a specific member of a net. When used in a function name, the `memNet` part implies the presence of a net ID and an integer index. The index specifies which bit of the net is under consideration, with index zero referring to the left-most bit.

Net Hierarchy

A net can also be hierarchical and have parent or child nets that represent the current status of the connectivity between these nets. The net hierarchy allows tools such as placement and routing and layout extraction to express a physically unrealized connection of two nets.

Parent and child relationships within a net hierarchy are conceptually different from parent and child relationships for figures. For details about parent and child relationships for figures, see [“CDBA Figure Functions”](#) on page 207.

The root net for a specified net is found by following the parent relationships upward until there is a net that has no parent. The root net for a net that has no parent is itself. Two nets are defined as logically equivalent if they have the same root net.

Net Names

A net name must be different from all other nets unless they have the same root net (that is, are logically equivalent). The names of all nets that share a common root net are aliases for that hierarchy of nets. The preferred name for that net is the name of the root net.

Connection Status

The status of connections can be retrieved from a net. The C data type for the constants is `dbConnStatusType`. The following table gives the values for connection status.

Connection Status (`dbConnStatusType`)

Constant	Description
<code>dbcConnected</code>	The connections of this net are physically implemented by the objects of this net (for example, abutting).
<code>dbcEdgeConnected</code>	The connections of this net are physically implemented, but one or more connections are made through edge-assigned pins.
<code>dbcNeedToConnect</code>	The connections of this net are not physically implemented and need to be.
<code>dbcConnectOutside</code>	One or more of the connections of this net are not implemented and need to be connected above them in the hierarchy to implement them.

If a net is connected to one or more pins, it must have a terminal. Pins can connect to child nets of a specified net. However, a terminal must always refer to the root net for the child nets because the terminal can point to only one net.

The net connected to an instance pin can be either the same as, or a child of, the net connected to the corresponding instance terminal. An instance terminal must be logically connected to a net prior to connecting child nets to its instance pins.

Functions Acting on Nets

The following functions act on nets.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateNet

```
dbNetId  
dbCreateNet(  
    dbCellViewId cellViewId, String netName, dbNetId parentNetId  
);
```

Creates a new net in a cellview. If the net already exists, returns `NULL`.

To create a child net, the *parentNetId* object must not be `nil`. If *netName* is an empty string or `nil`, the name of the parent net is used. Also, the child net will have same number of bits as the parent net. If *netName* is specified, then it must imply the same number of bits as the parent net. Child nets do not have unique names in CDBA. A child net is always created even when other nets exist with the same name.

dbCreateNamedSubNet

```
dbNetId  
dbCreateNamedSubNet(  
    dbNetId    parentNetId,  
    const char *subNetName  
);
```

Description

Creates a subnet with the specified name. The behavior is different from [dbCreateNet](#) because when a root net ID is specified, `dbCreateNameSubNet` creates a named `oaScalarNet` with the subnet name and associates it with the parent net.

The following restrictions apply:

- The specified *parentNetId* must be a single-bit net.
- The *parentNetId* must be a scalar net, which means that it cannot be a bus-bit name.
- The *parentNetId* cannot have any subnets with the specified subnet name.
- The *subNetName* must imply that it is a bit net.
- The block cannot have any nets with the specified subnet name.

Arguments

<i>parentNetId</i>	Database ID of the root net with which the subnet will be associated. The <i>parentNetId</i> also provides the cellview in which the new net is created.
<i>subNetName</i>	Name for the new subnet.

Example

In the following example, generating nets in the cellview `cvId` creates the named nets `a` and `sn2`. Generating the subnets of net `a` creates named subnets `sn1` and `sn2`.

```
dbNetId    rootNet = dbCreateNet(cvId, "a");
```

Create subnet `sn1`.

```
dbNetId    subNet1 = dbCreateNet(cvId, "sn1", rootNet);
```

Create subnet `sn2`.

```
dbNetId    subNet2 = dbCreateNamedSubNet(rootNet, "sn2");
```

dbMakeNet

```
dbNetId
dbMakeNet (
    dbCellViewId cellViewId, String netName, dbNetId parentNetId
);
```

These two functions create a net by name. The net name can imply that the net contains one or more bits. If the `parentNetId` is NULL and a topmost parent net of the specified name already exists, `dbCreateNet` returns NULL and `dbMakeNet` returns the net.

If the `parentNetId` is not NULL, the newly created net is placed as a child of the specified parent net. Child and parent nets can have different names, but they must imply the same number of bits. If the name for a child net is specified as NULL or an empty string, the name of the parent is used. Child and parent nets always refer to the same signals, but only topmost parent nets are listed as member nets and aliases of a signal.

dbCreateUniqueNamedNet

```
dbNetId
dbCreateUniqueNamedNet (
    dbCellViewId cellViewId, String prefix, int numBits
);
```

Creates a new net with a unique name. If `numBits` is one, the name is made of `prefix` suffixed by a unique number. If `numBits` is greater than one, the name is further suffixed by `<0:numBits-1>`. If `numBits` is less than negative one, the name is suffixed by `<-numBits-1:0>`.

Examples

```
dbCreateUniqueNamedNet(cvId, "N", 4) ==> N<0:3>
dbCreateUniqueNamedNet(cvId, "N", -4) ==> N<3:0>
```

dbDeleteNet

```
Boolean
dbDeleteNet (
    dbNetId netId
);
```

Deletes the specified net.

dbDeleteEmptyNet

Boolean

```
dbDeleteEmptyNet(  
    dbNetId netId  
);
```

Deletes the net if the net does not have any figures, instance terminals, terminals, instance pins, pins, or children. The function returns TRUE if the net is successfully deleted. Otherwise, the function returns FALSE.

dbGetNetName

String

```
dbGetNetName(  
    dbNetId netId  
);
```

Gets the name of this net.

dbGetNetNumBit

int

```
dbGetNetNumBit(  
    dbNetId netId  
);
```

Gets the total number of bits in this net.

dbGetMemNetName

String

```
dbGetMemNetName(  
    dbNetId netId, int memIndex  
);
```

Gets the name for this member net.

dbGetMemNetSig

dbSigId

```
dbGetMemNetSig(  
    dbNetId netId, int memIndex  
);
```

Gets the signal associated with this (bit) member net.

dbGetNetTerm

```
dbTermId  
dbGetNetTerm(  
    dbNetId netId  
);
```

Gets the terminal connected to this net, if any.

dbGetNetParent

```
dbNetId  
dbGetNetParent(  
    dbNetId netId  
);
```

Gets the parent net, if any.

dbGetNetRootNet

```
dbNetId  
dbGetNetRootNet(  
    dbNetId netId  
);
```

Gets the topmost parent net of this net. Returns the same net if it has no parent.

dbIsNetVoltageRangeSet

Boolean

```
dbIsNetVoltageRangeSet (  
    dbNetId netId  
);
```

Description

Indicates whether a voltage range has been set on the net or its block.

Arguments

<i>netId</i>	Database ID of a net.

Return Value

Boolean	TRUE if a voltage range has been set on the net or its block. FALSE if no voltage range has been set on the net and its block.

dbGetNetVoltageRangeOnNet

Boolean

```
dbGetNetVoltageRangeOnNet (  
    dbNetId    netId,  
    float      *minVoltage,  
    float      *maxVoltage  
);
```

Description

Returns the min and max voltages on a net without inheriting it from the block.

Arguments

<i>netId</i>	Database ID of a net.
<i>minVoltage</i>	The minimum voltage value retrieved from the specified net ID.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

maxVoltage The maximum voltage value retrieved from the specified net ID.

Return Value

Boolean TRUE if minimum and maximum voltage values are retrieved successfully.
FALSE if the operation fails.

Example

```
dbGetNetVoltageRangeOnNet(netId, minVolt, maxVolt)  
=> true
```

dbIsNetVoltageRangeSetOnNet

Boolean

```
dbIsNetVoltageRangeSetOnNet(  
    dbnetId netId  
);
```

Description

Indicates whether a voltage range has been set on the specified net.

Arguments

netId Database ID of a net.

Return Value

Boolean TRUE if a voltage range has been set on the specified net.
FALSE if no voltage range has been set on the specified net.

dbGetNetVoltageRangeSource

```
dbNetVoltageRangeSource  
dbGetNetVoltageRangeSource (  
    dbNetId netId  
);
```

Description

Returns the voltage range source of the specified net ID.

Arguments

<i>netId</i>	The net ID of the voltage range source that must be returned.

Return Value

<i>dbNetVoltageRangeSource</i>	A text string representing the voltage range source of the specified net ID.

dbSetNetVoltageRangeSource

```
Boolean  
dbSetNetVoltageRangeSource (  
    dbNetId      netId,  
    dbNetVoltageRangeSource source  
);
```

Description

Sets the voltage range source on the specified net ID.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>netId</i>	The net ID whose voltage range source must be set.
<i>source</i>	The voltage range source to be set on the specified net ID.

Return Value

<i>Boolean</i>	TRUE, if the voltage range source was set successfully on the specified net ID. FALSE, if the voltage range source was not set on the specified net ID.

dbUnsetNetVoltageRangeSource

Boolean

```
dbUnsetNetVoltageRangeSource(  
    dbNetId netId  
);
```

Description

Resets the voltage range source of the specified net ID to its default value.

Arguments

<i>netId</i>	The net ID whose voltage range source must be set to its default value.

Return Value

<i>Boolean</i>	TRUE, if the default voltage range source was reset successfully on the specified net ID. FALSE, if the default voltage range source was not reset on the specified net ID.

dbIsNetOptical

Boolean

```
dbIsNetOptical(  
    dbNetId netId  
);
```

Description

Checks whether the specified net contains optical data.

Arguments

<i>dbNetId</i>	The database ID of a net.

Return value

<i>Boolean</i>	TRUE, if the net contains optical data. FALSE, if the net does not contain optical data.

Example

```
dbIsNetOptical (netId);
```

dbMoveChildNet

Boolean

```
dbMoveChildNet(  
    dbNetId childNetId, dbNetId parentNetId  
);
```

Moves the child net to a new parent. The old parent must exist. Only child nets can be moved.

dbMergeNet

dbNetId

```
dbMergeNet(  
    dbNetId net1, dbNetId net2  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Merges *net2* into *net1* and deletes *net2*. Merging refers to putting *net2*'s connections, figures, and member nets of the signals onto *net1*. Both nets must have the same number of bits. This is useful when a geometry is added that causes two different nets to become shorted. All of the operations of a merge can be done with separate database function calls, but *dbMergeNet* is more efficient.

dbNetHasGlobal

Boolean

```
dbNetHasGlobal(  
    dbNetId netId  
);
```

Returns TRUE if at least one signal in the net is global. Otherwise, the function returns FALSE.

dbNetHasFig

Boolean

```
dbNetHasFig(  
    dbNetId netId  
);
```

Returns TRUE if the net has at least one figure. Otherwise, the function returns FALSE.

dbNetHasTerm

Boolean

```
dbNetHasTerm(  
    dbNetId netId  
);
```

Returns TRUE if the net is connected to at least one terminal. Otherwise, the function returns FALSE.

dbNetHasInstTerm

Boolean

```
dbNetHasInstTerm(  
    dbNetId netId  
);
```


Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Returns TRUE if the net is connected to at least one instance terminal. Otherwise, the function returns FALSE.

dbNetHasPin

Boolean

```
dbNetHasPin(  
    dbNetId netId  
);
```

Returns TRUE if the net is connected to at least one pin. Otherwise, the function returns FALSE.

dbNetHasChild

Boolean

```
dbNetHasChild(  
    dbNetId netId  
);
```

Returns TRUE if the net has a child net. Otherwise, the function returns FALSE.

dbGetNetPriority

int

```
dbGetNetPriority(  
    dbNetId netId  
);
```

Gets the priority of the net. The priority must be an integer between 0 and 127.

dbSetNetPriority

Boolean

```
dbSetNetPriority(  
    dbNetId netId, int priority  
);
```

Sets the priority of the net. The priority must be an integer between 0 and 127.

dbGetNetConnStatus

dbConnStatusType

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetNetConnStatus(  
    dbNetId netId  
);
```

Gets the connectivity status of this net's connections. The table [“Connection Status \(dbConnStatusType\)”](#) on page 685 gives the values for connection status.

dbSetNetConnStatus

Boolean

```
dbGetNetConnStatus(  
    dbNetId netId, dbConnStatusType connStatus  
);
```

Sets the connectivity status of this net's connections. The table [“Connection Status \(dbConnStatusType\)”](#) on page 685 gives the values for connection status.

dbGetNetPowerDomain

Boolean

```
dbGetNetPowerDomain(  
    dbNetId netId,  
    unsigned int *numDomains,  
    String **domains  
);
```

Description

Returns the power domain attribute of a net. The power domain strings are set to the array for which memory is allocated by this function.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Arguments

<i>netId</i>	The database ID of the net object.
<i>numDomains</i>	Number of power domain strings in the array.
<i>domains</i>	An array of power domain strings.

Return Values

Boolean	Returns TRUE if the power domain attribute is successfully returned. Returns FALSE otherwise.

dbSetNetPowerDomain

Boolean

```
dbSetNetPowerDomain(  
    dbNetId netId,  
    unsigned int numDomains,  
    String *domains  
);
```

Description

Sets the power domain attribute of a net.

Arguments

<i>netId</i>	The database ID of the net object.
<i>numDomains</i>	Number of power domain strings in the array.
<i>domains</i>	An array of power domain strings.

Return Values

Boolean	Returns TRUE if the power domain attribute is successfully set. Returns FALSE otherwise.

dbStartGenNetToSig

dbGenStateId

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbStartGenNetToSig(  
    dbNetId netId  
);
```

Starts the generator for signals in this net, ordered from left-most to right-most bit. Note the same signal can occur more than once in a net.

dbGenNetToSig

```
Boolean  
dbGenNetToSig(  
    dbGenStateId state, dbSigId pSigId  
);
```

Generates signals in this net, ordered from left-most to right-most bit. Note the same signal can occur more than once in a net.

dbStartGenNetToFig

```
dbGenStateId  
dbStartGenNetToFig(  
    dbNetId netId  
);
```

Starts the generator for shapes or instances in this net, if any.

dbGenNetToFig

```
Boolean  
dbGenNetToFig(  
    dbGenStateId state, dbFigId pFigId  
);
```

Generates shapes or instances in this net, if any.

dbStartGenNetToInstTerm

```
dbGenStateId  
dbStartGenNetToInstTerm(  
    dbNetId netId  
);
```

Starts the generator for instance terminals connected to this net, if any.

dbGenNetToInstTerm

Boolean

```
dbGenNetToInstTerm(  
    dbGenStateId state, dbInstTermId pInstTermId  
);
```

Generates instance terminals connected to this net, if any.

dbStartGenNetToPin

dbGenStateId

```
dbStartGenNetToPin(  
    dbNetId netId  
);
```

Starts the generator for pins connected to this net, if any.

dbGenNetToPin

Boolean

```
dbGenNetToPin(  
    dbGenStateId state, dbPinId pPinId  
);
```

Generates pins connected to this net, if any.

dbStartGenNetToAllInstTerm

dbGenStateId

```
dbStartGenNetToAllInstTerm(  
    dbNetId netId  
);
```

Starts the generator for instance terminals connected to this net and all of its child nets. The net hierarchy is traversed in a pre-ordering method.

dbGenNetToAllInstTerm

Boolean

```
dbGenNetToAllInstTerm(  
    dbGenStateId state, dbInstTermId pInstTermId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Generates instance terminals connected to this net and all of its child nets. The net hierarchy is traversed in a pre-ordering method.

dbStartGenNetToChild

```
dbGenStateId  
dbStartGenNetToChild(  
    dbNetId netId  
); Boolean
```

Starts the generator for child nets one level below this net, if any.

dbGenNetToChild

```
Boolean  
dbGenNetToChild(  
    dbGenStateId state, dbNetId pNetId  
);
```

Generates child nets one level below this net, if any.

dbStartGenNetToAllChild

```
dbGenStateId  
dbStartGenNetToAllChild(  
    dbNetId netId  
);
```

Starts the generator for all child nets under the specified net using a post-ordering traversal (starting with the bottom level children up to the parent).

dbGenNetToAllChild

```
Boolean  
dbGenNetToAllChild(  
    dbGenStateId state, dbNetId pNetId  
);
```

Generates all child nets under the specified net using a post-ordering traversal, starting with the bottom level children up to the parent.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetNetRoutePattern

Boolean

```
dbGetNetRoutePattern(  
    dbNetId          netId,  
    dbRoutePattern*  routePattern  
);
```

Description

Populates the `routePattern` parameter with the current `routePattern` being used for the net.

Arguments

<i>netId</i>	The database ID of the net object.
<i>routePattern</i>	The pointer to the <i>dbRoutePattern</i> data to be populated.

Return Values

Boolean	Returns TRUE if the <code>routePattern</code> was successfully populated. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetNetRoutePattern

Boolean

```
dbSetNetRoutePattern(  
    dbNetId          netId,  
    dbRoutePattern   routePattern  
);
```

Description

Sets the routePattern for the specified net to the value specified with the routePattern parameter.

Arguments

<i>netId</i>	The database ID of the net object.
<i>routePattern</i>	The value to be set for the net's routePattern.

Return Values

Boolean	Returns TRUE if the routePattern for the specified net is successfully set. Returns FALSE otherwise.

Signals

A signal is a database object that ties together all nets that implement the signal. Therefore, a signal can belong to several different nets. You can use a signal when netlisting to visit all the signal's connections at a given hierarchy level. A signal is represented in a name by the `sig` part.

When you create or delete nets, signals are automatically created or deleted as well. Each signal has a unique name derived from one of the top-level nets that refers to the signal. Two distinct signals can be merged into one, creating a new alias for the surviving signal. This new alias refers to the merged signal.

Hierarchy for Signals

Parent and child nets refer to the same signals. Parent and child relationships for nets are conceptually different from parent and child relationships for figures. For details about parent and child relationships for figures, see [Chapter 9, “CDBA Parent-Child Relationship Functions,”](#).

A global signal connects to other signals with the same name elsewhere in the hierarchy without requiring an explicit connection through the hierarchy. By convention, in Cadence® schematics (interpreted by the schematic extractor), a global signal's name ends with an exclamation point (!), such as *gnd!*.

Signal Types

The following table gives information on signal types. `dbSigType` is the C data type for signal type constants.

Signal Types (dbSigType)

Constant	Description
<code>dbcSignal</code>	Carries an ordinary signal.
<code>dbcGround</code>	Carries the reference signal for the cellview.
<code>dbcSupply</code>	Provides power for the cellview.
<code>dbcClock</code>	Carries timing-critical clocking for the cellview.
<code>dbcTestLatch</code>	Carries part of a test scan chain.

Functions Acting on Signals

The following functions retrieve information about and act on signals.

dbGetSigName

```
String  
dbGetSigName(  
    dbSigId sigId  
);
```

Returns the name for this signal as a permanent string. The system creates the signal when you create a net carrying that signal. The name of the signal is set to the one implied by that net. This signal can have aliases after it is merged with other signals.

dbSetSigName

```
Boolean  
dbSetSigName(  
    dbSigId sigId, String name  
);
```

Sets the name of the specified signal to one of its aliases. The specified alias must be the name of one of the signal's member nets, and the signal name must be valid and in a form required by the rules.

dbIsSigGlobal

```
Boolean  
dbIsSigGlobal(  
    dbSigId sigId  
);
```

Returns TRUE if the signal has a global attribute set.

dbIsSigOptical

```
Boolean  
dbIsSigOptical(  
    dbSigId sigId  
);
```

Description

Checks whether the specified signal contains is optical data.

Arguments

<i>sigId</i>	The database ID of a signal type.

Return value

<i>Boolean</i>	TRUE, if the signal has optical data. FALSE, if the signal does not contain optical data.

Example

```
dbIsSigOptical(sigId);
```

dbSetSigGlobal

```
Boolean  
dbSetSigGlobal(  
    dbSigId sigId  
);
```

Sets global attributes for the signal to TRUE.

dbResetSigGlobal

```
Boolean  
dbResetSigGlobal(  
    dbSigId sigId  
);
```

Resets global attributes for the signal to FALSE.

dbStartGenSigToAlias

```
dbGenStateId  
dbStartGenSigToAlias(  
    dbSigId sigId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Starts the generator for alternative names for this signal. Currently, the names generated include the name of every member net that implements this signal.

dbGenSigToAlias

Boolean

```
dbGenSigToAlias(  
    dbGenStateId state, String pAlias  
);
```

Generates alternative names for this signal. Currently the names generated include the name of every member net that implements this signal.

dbGetSigType

dbSigType

```
dbGetSigType(  
    dbSigId sigId  
);
```

Starts the generator to get or set the type of this signal.

dbSetSigType

Boolean

```
dbSetSigType(  
    dbSigId sigId, dbSigType sigType  
);
```

These two functions get or set the type of this signal. The following table lists valid signal types.

Signal Types (dbSigType)

Constant	Description
dbcSignal	Is an ordinary signal.
dbcGround	Is the reference signal for the cellview.
dbcSupply	Provides power for the cellview.
dbcClock	Carries timing critical clocking.
dbcTestLatch	Is part of a test scan chain.

dbStartGenSigToMemNet

```
dbGenStateId  
dbStartGenSigToMemNet(  
    dbSigId sigId  
);
```

Starts the generator for member nets that implement this signal.

dbGenSigToMemNet

```
Boolean  
dbGenSigToMemNet(  
    dbGenStateId state, dbNetId pNetId, int pMemIndex)
```

Generates member nets that implement this signal.

dbStartGenSigToMemTerm

```
dbGenStateId  
dbStartGenSigToMemTerm(  
    dbSigId sigId  
);
```

Starts the generator for member terminals connected to this signal through its nets.

dbGenSigToMemTerm

```
Boolean  
dbGenSigToMemTerm(  
    dbGenStateId state, dbTermId pTermId, int pMemIndex  
);
```

Generates member terminals connected to this signal through its nets.

dbStartGenSigToMemInstTerm

```
dbGenStateId  
dbStartGenSigToMemInstTerm(  
    dbSigId sigId  
);
```

Starts the generator for member instance terminals connected to this signal.

dbGenSigToMemInstTerm

Boolean

```
dbGenSigToMemInstTerm(  
    dbGenStateId state, dbInstTermId pInstTermId, int pMemIndex  
);
```

Generates member instance terminals connected to this signal.

dbMergeSignal

dbSigId

```
dbMergeSignal(  
    dbSigId sigId1, dbSigId sigId2  
);
```

Merges two signals. Signal `sigId1` survives the merge. Signal `sigId2` is deleted after the merge. Also merges member nets of `sigId2`. After the merge, `sigId1` can be referred to by the name and aliases of `sigId2`. If either signal is global, the merged signal is global. The merge is not allowed if each signal is already connected to a member terminal.

Terminals

A terminal is a database object that makes a net available for connection on instances in a cellview. Nets must have terminals to make connections using an instance of the cellview. This requirement implies that global signals must have a net and a terminal. The system currently does not enforce this relationship. A terminal is represented in a name by the `term` part.

Member Terminals

A member terminal refers to a specific member of a terminal. A member terminal is represented in a name by the `memTerm` part. When used in a function name, `memTerm` implies the presence of a `termId` and an integer index. The index specifies which bit of the terminal is of interest, with index zero referring to the left-most bit.

Terminal Names

Each terminal has a unique name that is usually the same as the net to which it connects (but it can be different). A member terminal with the same name can occur more than once within the same or distinct terminal. For example, a terminal named A, A, B can coexist within the same cellview as another terminal named B, A. This feature implies that a signal within a cellview can be accessible through one or many member terminals of the same name.

Terminal Connections

A terminal must contain only one top-level net and can contain pins. The pins of a terminal can be connected to either a top-level net or to a child of the top-level net, but the terminal for those pins can only connect to a top-level net.

Terminal Direction

A terminal has a direction attribute that specifies the drive direction for the terminal as viewed from outside the cellview. The following table illustrates the terminal directions associated with the `dbTermDirectType` C data type.

Terminal Direction (`dbTermDirectType`)

Constant	Description
<code>dbcInput</code>	Signals flow into the instance. Can have a value assigned to it from above in the hierarchy.
<code>dbcOutput</code>	Signals flow out of the instance.
<code>dbcInputOutput</code>	Signals flow out of and into the instance, but at different times. Can be either input or output (but not both simultaneously).
<code>dbcSwitch</code>	Signals flow out of and into the instance, both possibly at the same time (as in a bidirectional switch). Can simultaneously be both input and output.
<code>dbcJumper</code>	Signals can flow through the cellview but are not used within the cellview. Can only be used above in the hierarchy.
<code>dbcUnused</code>	This terminal is not used and must not be used. Must not be used in this cellview, below or above.
<code>dbcTriState</code>	In addition to being able to drive a logical 0 or a 1, this terminal is able to assume a high impedance state effectively removing the terminal from an active connection.

You can make visible on the terminal other attributes of the net that the terminal represents.

Terminal Position

You can create and reference terminals by position. Nets can connect to an instance terminal that is bound to a terminal position in the master rather than being bound to a terminal by name. You can set and query terminal position, find a terminal by position, and create an `instTerm` by position.

- Positions are numbered from 0 to n , where 0 corresponds to the first position in a module declaration.
- Assigning a terminal to a taken position is an error.
- Resetting a terminal assigned to a position is allowed.

Functions Acting on Terminals

The following functions retrieve information about or act on terminals.

dbCreateTerm

```
dbTermId  
dbCreateTerm(  
    dbNetId netId, String termName, dbTermDirectType direction, int nullInt  
);
```

Creates a terminal for a net. `termName` can be NULL, indicating that the name of the net is copied onto the terminal. The terminal name, if specified, must imply the number of bits equal to the number of bits in the net. `direction` is one of the values listed in the previous table.

A terminal can be created for a net that has signals already attached to other terminals. This means that one signal can be connected to more than one terminal through its nets. However, all member terminals connected to the same signal must have the same name.

dbCreateBlockTerm

```
dbTermId  
dbCreateBlockTerm(  
    dbNetId          netId,  
    const char       *termName,  
    dbTermDirectType direction,  
    Boolean          physOnly  
);
```

Creates a terminal for the net and specifies its visibility in the module (logical) domain.

The Embedded Module Hierarchy (EMH) status of the terminal is specified through `physOnly`. TRUE creates a terminal that is physical only. False creates a terminal that is both logical and physical.

dbDeleteTerm

```
Boolean  
dbDeleteTerm(  
    dbTermId termId  
);
```

Deletes the specified terminal.

dbGetTermDirect

```
dbTermDirectType  
dbGetTermDirect(  
    dbTermId termId  
);
```

Returns the drive direction for the terminal.

dbSetTermDirect

```
Boolean  
dbSetTermDirect(  
    dbTermId termId, dbTermDirectType direction  
);
```

Sets the drive direction for the terminal.

dbSetTermName

```
Boolean  
dbSetTermName(  
    dbTermId termId, String newTermName  
);
```

Modifies the terminal name. The number of bits implied by the new name must equal the number of bits in the net of the terminal.

If the new terminal type is different from the old terminal type, a new terminal is created. While doing so, the prop and pin information of the old terminal is preserved. Also, if the terminal has a textdisplay associated with it, a new textDisplay with the attributes of the original textDisplay is created on the new terminal, and the original textDisplay is deleted.

dbGetTermName

```
String  
dbGetTermName(  
    dbTermId termId  
);
```

Returns the terminal name.

dbGetMemTermName

String

```
dbGetMemTermName (
    dbTermId termId, int memIndex
);
```

Returns the member terminal name.

dbGetTermNet

dbNetId

```
dbGetTermNet (
    dbTermId termId
);
```

Returns the net to which this terminal connects internally.

dbGetTermNumBit

int

```
dbGetTermNumBit (
    dbTermId termId
);
```

Returns the number of bits in this terminal. This is equivalent to asking for the number of bits in the corresponding net.

dbTermHasPin

Boolean

```
dbTermHasPin (
    dbTermId termId
);
```

Returns TRUE if this terminal has at least one pin.

dbStartGenTermToPin

dbGenStateId

```
dbStartGenTermToPin (
    dbTermId termId
);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Starts the generator for pins attached to this terminal, if any.

dbGenTermToPin

Boolean

```
dbGenTermToPin(  
    dbGenStateId state, dbPinId pPinId  
);
```

Generates pins attached to this terminal, if any.

dbGetTermRouteMethod

Boolean

```
dbGetTermRouteMethod(  
    dbTermId          termId,  
    dbRouteMethod*    routeMethod  
);
```

Description

Returns the routeMethod of the specified Term.

Arguments

<i>termId</i>	The database ID of the Term object.
<i>routeMethod</i>	Returns the routeMethod of the term.

Return Values

Boolean	Returns TRUE if the routeMethod was found. Returns FALSE otherwise.

dbSetTermRouteMethod

Boolean

```
dbSetTermRouteMethod(  
    dbTermId          termId,  
    dbRouteMethod     routeMethod  
);
```

Description

Sets the routeMethod for the specified term to the value specified with the routeMethod parameter.

Arguments

<i>termId</i>	The database ID of the term object.
<i>routeMethod</i>	The value to be set for the routeMethod.

Return Values

Boolean	Returns TRUE if the routeMethod for the specified term is successfully set. Returns FALSE otherwise.

dbGetTermPosition

```
unsigned int  
dbGetTermPosition(  
    dbTermId      termId  
);
```

Description

Gets the position assigned to a specified terminal.

Arguments

<i>termId</i>	The database ID of the Term object.

Return Values

<i>unsigned int</i>	Returns the position assigned to the terminal. Positions are numbered from 0 to n , where 0 corresponds to the first position in a module declaration. Returns dbcNullIndex if the terminal has no position assigned.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbSetTermPosition

Boolean

```
dbGetTermPosition(  
    dbTermId          termId,  
    unsigned int      position  
);
```

Description

Assigns or changes the position of a specific terminal. A multi-bit terminal occupies a position similar to a scalar terminal.

Arguments

<i>termId</i>	The database ID of the term object.
<i>position</i>	The position to which to assign the terminal. The position can be from 0 to <i>n</i> , where 0 is the first position.

Return Values

unsigned int	Returns TRUE if the position is assigned. Returns FALSE if the position is already assigned to another terminal or if a negative position is specified.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbUnsetTermPosition

```
void  
dbUnsetTermPosition(  
    dbTermId      term  
);
```

Description

Clears the position from the specified terminal.

Arguments

<i>term</i>	The database ID of the Term object.

Return Values

<i>void</i>	Returns a void if the position is cleared. Nothing is done if the terminal has no position assigned.

dbGetTermsWithPosition

Boolean

```
dbGetTermsWithPosition(  
    dbCellViewId      cvId,  
    unsigned int       *maxPosition,  
    dbTermId           **termIds  
);
```

Description

Gets a collection of terminals in the cellview that have positions assigned.

Arguments

<i>cvId</i>	The database ID of the cellview.
<i>maxPosition</i>	Returns the highest assigned position in the cellview. The position number is also the maximum index in the array.
<i>termIds</i>	Returns an array of terminals in the cellview that have assigned positions. The position of a terminal in the array reflects its assigned position. The array can contain a NULL <i>dbTermId</i> if a specific position has no terminal assigned.

Return Values

Boolean	Returns TRUE if the cellview contains at least one terminal with an assigned position. Returns FALSE if no terminals have assigned positions.

dbFindTermByPosition

```
dbTermId  
dbFindTermByPosition(  
    dbCellViewId      cvId,  
    unsigned int       position  
);
```

Description

Searches the cellview for a terminal assigned to the specified position.

Arguments

<i>cvId</i>	The cellview to search.
<i>position</i>	The position at which to find the terminal.

Return Values

<i>dbTermId</i>	Returns the terminal. Returns a dbcNullTermId if no terminal is found.

Functions Acting on Terminals

A `softConnect` terminal is a terminal that does not draw or carry any current.

A typical four terminal device has terminals for gate, drain, source, and the bulk node. The bulk node terminal is often not shown in a schematic representation. If the bulk node is shown, it is shown as connected to either power or ground. Similarly in a layout representation, the bulk node terminal is connected to either power or ground, but electrically it does not draw any current. Hence, it can be designated as a `softConnect` terminal.

Terminals are either explicit or implicit. Explicit terminals are explicitly created and managed by the user, whereas, implicit terminals are created and managed automatically by the database. An example of an implicit terminal would be one of the bit terminals associated with an explicit, multi-bit terminal.

Note: Only explicit, single-bit terminals can be designated as `softConnect` terminals.

You can use the `dbSetTermSoftConnect` function to designate a terminal as a `softConnect` terminal.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

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`.

The following functions retrieve information about or act on terminals.

dbSetTermSoftConnect

```
void
dbSetTermSoftConnect (
    dbTermId      term
    Boolean        value
);
```

Sets the `softConnect` attribute of the specified terminal to the given value.

The terminal must be an explicit, single-bit terminal. If you set the `softConnect` attribute on an implicit or multi-bit terminal, an error is reported.

If the `softConnect` attribute of the terminal is set to `FALSE`, the associated `pinlessTerminalLayer` and `connectToLayer` attributes are removed (if set).

dbIsTermSoftConnect

```
Boolean
dbIsTermSoftConnect (
    dbTermId      term
);
```

Checks the `softConnect` attribute value of the specified terminal

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

Since the `softConnect` attribute can only be set on explicit, single-bit terminals, using this query on an implicit or multi-bit terminal always returns `FALSE`.

dbSetSoftConnectTermPinlessLayer

```
void
dbSetSoftConnectTermPinlessLayer(
    dbTermId      term
    dbLayer        pinlessTermLayer
);
```

Sets the `pinlessTerminalLayer` attribute of the specified `softConnect` terminal.

If you set this layer attribute on a terminal that is not designated as `softConnect`, an error is reported.

dbUnsetSoftConnectTermPinlessLayer

```
void
dbUnsetSoftConnectTermPinlessLayer(
    dbTermId      term
);
```

Removes the `pinlessTerminalLayer` attribute set on a terminal designated as `softConnect`.

dbGetSoftConnectTermPinlessLayer

```
dbLayer
dbGetSoftConnectTermPinlessLayer(
    dbTermId      term
);
```

Returns the `pinlessTerminalLayer` of the `softConnect` terminal as a `dbLayer`. If the `pinlessTerminalLayer` is not set for this terminal, an error is reported and `dbcNullLayer` is returned.

dbHasSoftConnectTermPinlessLayer

```
Boolean
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbHasSoftConnectTermPinlessLayer(  
    dbTermId          term  
);
```

Checks if the given `softConnect` terminal has the `pinlessTerminalLayer` attribute set. If the given terminal is not a `softConnect` terminal, `FALSE` is returned.

dbSetSoftConnectTermConnectToLayer

```
void  
  
dbSetSoftConnectTermConnectToLayer(  
    dbTermId          term  
    dbLayer           connectToLayer  
);
```

Sets the `connectToLayer` attribute of the specified `softConnect` terminal.

If you set this layer attribute on a terminal that is not designated as `softConnect`, an error is reported.

dbGetSoftConnectTermConnectToLayer

```
dbLayer  
  
dbGetSoftConnectTermConnectToLayer(  
    dbTermId          term  
);
```

Returns the `connectToLayer` of the `softConnect` terminal as a `dbLayer`. If the `connectToLayer` is not set for this terminal, an error is reported and `dbcNullLayer` is returned.

dbHasSoftConnectTermConnectToLayer

```
Boolean  
  
dbHasSoftConnectTermConnectToLayer(  
    dbTermId          term  
);
```

Checks if the given `softConnect` terminal has the `connectToLayer` attribute set. If the given terminal is not a `softConnect` terminal, `FALSE` is returned.

dbGetAnyTermGroundSensitivity

```
dbTermId  
dbGetAnyTermGroundSensitivity(  
    dbTermId termId  
);
```

Description

Returns ground sensitivity terminal information for the specified terminal. This terminal can be a multi-bit terminal. When it is a multi-bit terminal, the value returned represents the ground sensitivity terminal on the majority of the bits of the terminal.

Arguments

<i>termId</i>	Database ID of the specified terminal.

Value Returned

<i>dbTermId</i>	Database ID of the ground sensitivity terminal. Returns <code>NULL</code> if all or most of the bits of a multi-bit terminal do not have a ground sensitivity terminal set, or an error occurs.

Examples

Suppose `myTerm` is a bus terminal of name "A[0:3] ", and the ground sensitivity terminal on the four bits are "gnd, gnd, gnda, gnd".

```
dbTermId groundSensitivity = dbGetAnyTermGroundSensitivity(myTerm)
```

Returns "gnd" because it appears on three out of the four bits.

dbGetAnyTermSupplySensitivity

```
dbTermId  
dbGetAnyTermSupplySensitivity(  
    dbTermId termId  
);
```

Description

Returns supply sensitivity terminal information for the specified terminal. The terminal can be a multi-bit terminal. When it is a multi-bit terminal, the value returned represents the supply sensitivity terminal on the majority of the bits of the terminal.

Arguments

<i>termId</i>	Database ID of the specified terminal.

Value Returned

<i>dbTermId</i>	Database ID of the supply sensitivity terminal. Returns <code>NULL</code> if all or most of the bits of a multi-bit terminal do not have a supply sensitivity terminal set, or an error occurs.

Examples

Suppose `myTerm` is a bus terminal of name "A[0:3] ", and the supply sensitivity terminal on the four bits are "vdd, vdd, vdda, vdd".

```
dbTermId supplySensitivity = dbGetAnyTermSupplySensitivity(myTerm)
```

Returns "vdd" because it appears on three out of the four bits.

dbGetAnyTermGroundSensitivityAll

Boolean

```
dbGetAnyTermGroundSensitivityAll(  
    dbTermId termId,  
    unsigned int *numTerms,  
    dbTermId **groundSensitivityValues  
);
```

Description

Retrieves the ground sensitivity terminal on all bits of the specified terminal. This function fills the memory pointed by *numTerms* and *groundSensitivityValues*.

Arguments

<i>termId</i>	Database ID of the specified terminal.
<i>*numTerms</i>	The number of ground sensitivity terminals in the array.
<i>**groundSensitivityValues</i>	The array containing the ground sensitivity terminals.

Value Returned

<i>Boolean</i>	TRUE if the ground sensitivity terminal is retrieved successfully. FALSE if the <i>numTerms</i> and ground sensitivity values are not correctly filled. Check the Boolean value of this function before accessing <i>numTerms</i> and the ground sensitivity values.

Examples

Retrieves the ground sensitivity terminal on all bits of the terminal *myTerm*:

```
unsigned int  numTerms = 0;  
    dbTermId      *sensitivityTerms = NULL;  
Boolean res = dbGetAnyTermGroundSensitivityAll (myTerm, &numTerms,  
&sensitivityTerms);
```

Free the memory pointed by *sensitivityTerms* using:

```
voFreeArray(numTerms, sensitivityTerms, dbTermId);
```

dbGetAnyTermSupplySensitivityAll

Boolean

```
dbGetAnyTermSupplySensitivityAll(  
    dbTermId termId,  
    unsigned int *numTerms,  
    dbTermId **supplySensitivityValues  
);
```

Description

Retrieves the supply sensitivity terminal on all bits of the specified terminal. This function fills the memory pointed by *numTerms* and *groundSensitivityValues*.

Arguments

<i>termId</i>	Database ID of the specified terminal.
* <i>numTerms</i>	The number of supply sensitivity terminals in the array.
** <i>supplySensitivityValues</i>	The array containing the supply sensitivity terminals.

Value Returned

Boolean

TRUE if the supply sensitivity terminal is retrieved successfully.

FALSE if the *numTerms* and supply sensitivity values are not correctly filled.

Check the Boolean value of this function before accessing *numTerms* and the supply sensitivity values.

Example

Retrieves the supply sensitivity terminal on all bits of the terminal *myTerm*:

```
unsigned int  numTerms = 0;  
    dbTermId      *sensitivityTerms = NULL;  
Boolean res = dbGetAnyTermSupplySensitivityAll (myTerm, &numTerms,  
&sensitivityTerms);
```

Free the memory pointed by *sensitivityTerms* using:

```
voFreeArray(numTerms, sensitivityTerms, dbTermId);
```

dbIsAnyTermGroundSensitivityConsistent

Boolean

```
dbIsAnyTermGroundSensitivityConsistent(  
    dbTermId termId  
);
```

Description

Checks whether all the bits of the specified terminal have the same ground sensitivity terminal.

Argument

<i>dbTermId</i>	Database ID of the specified terminal.

Value Returned

<i>Boolean</i>	TRUE if all bits of the specified terminal have the same ground sensitivity term. FALSE if the bits of the specified terminal do not all have the same ground sensitivity term.

Example

Checks if all bits of the terminal `myTerm` have the same ground sensitivity terminal.

```
Boolean res = dbIsAnyTermGroundSensitivityConsistent(myTerm);
```

dbIsAnyTermSupplySensitivityConsistent

Boolean

```
dbIsAnyTermSupplySensitivityConsistent(  
    dbTermId termId  
);
```

Description

Checks whether all the bits of the specified terminal have the same supply sensitivity terminal.

Argument

<i>termId</i>	Database ID of the specified terminal.

Value Returned

<i>Boolean</i>	TRUE if all bits of the specified terminal have the same supply sensitivity term. FALSE if the bits of the specified terminal do not all have the same supply sensitivity term.

Example

Checks if all bits of the terminal `myTerm` have the same supply sensitivity terminal.

```
Boolean res = dbIsAnyTermSupplySensitivityConsistent(myTerm);
```

dbSetAnyTermGroundSensitivity

Boolean

```
dbSetAnyTermGroundSensitivity(  
    dbTermId termId,  
    dbTermId groundTermId  
);
```

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>termId</i>	Database ID of the specified terminal.
<i>groundTermId</i>	Database ID of the ground sensitivity terminal.

Value Returned

<i>Boolean</i>	TRUE if the ground sensitivity value is successfully set for the specified terminal. FALSE if the ground sensitivity value is not successfully 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

Suppose that the bus terminal `myTerm` named `A[0:3]`, `A[1]` is an explicit bit terminal.

```
Boolean res = dbSetAnyTermGroundSensitivity(myTerm, groundTermId);
```

Sets the ground sensitivity of `myTerm` with a term "gnd" for `A[0]`, `A[2]` and `A[3]`.

dbSetAnyTermSupplySensitivity

Boolean

```
dbSetAnyTermSupplySensitivity(  
    dbTermId termId,  
    dbTermId supplyTermId  
);
```

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>termId</i>	Database ID of the specified terminal.
<i>supplyTermId</i>	Database ID of the supply sensitivity terminal.

Value Returned

<i>Boolean</i>	TRUE if the supply sensitivity value is successfully set for the specified terminal. FALSE if the supply sensitivity value is not successfully set for the specified terminal. This may be because the 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

Suppose that the bus terminal `myTerm` named `A[0:3]`, `A[1]` is an explicit bit terminal.

```
Boolean res = dbSetAnyTermSupplySensitivity(myTerm, supplyTermId);
```

Sets the supply sensitivity of `myTerm` with a term `"gnd"` for `A[0]`, `A[2]` and `A[3]`.

Terminal Antenna Functions

```
#define dbcMaxAntennaModel    4
#define dbcMaxAntennaData    8
```

Antenna model names

```
constString dbvAntennaModelString[] = {
    "defaultAntennaModel",
    "secondAntennaModel",
    "thirdAntennaModel",
    "fourthAntennaModel"
};
```

Model enums

```
typedef enum dbAntennaModelType {
    dbcDefaultAntennaModel    = oacDefaultAntennaModel,
    dbcSecondAntennaModel     = oacSecondAntennaModel,
    dbcThirdAntennaModel      = oacThirdAntennaModel,
    dbcFourthAntennaModel     = oacFourthAntennaModel
};
```

Antenna data names

```
constString dbvAntennaDataString[] = {
    "ANTENNAGATEAREA",
    "ANTENNADIFFAREA",
    "ANTENNAPARTIALMETALAREA",
    "ANTENNAPARTIALMETALSIDEAREA",
    "ANTENNAPARTIALCUTAREA",
    "ANTENNAMAXAREACAR",
    "ANTENNAMAXSIDEAREACAR",
    "ANTENNAMAXCUTCAR"
};
```

Antenna data enums

```
typedef enum dbAntennaDataType {
    dbcAntennaGateArea          = 1, // "ANTENNAGATEAREA",
    dbcAntennaDiffArea          = 2, // "ANTENNADIFFAREA",
    dbcAntennaPartialMetalArea  = 3, // "ANTENNAPARTIALMETALAREA",
    dbcAntennaPartialMetalSideArea = 4, // "ANTENNAPARTIALMETALSIDEAREA",
    dbcAntennaPartialCutArea    = 5, // "ANTENNAPARTIALCUTAREA",
    dbcAntennaMaxAreaCAR        = 6, // "ANTENNAMAXAREACAR",
    dbcAntennaMaxSideAreaCAR    = 7, // "ANTENNAMAXSIDEAREACAR",
    dbcAntennaMaxCutCAR         = 8, // "ANTENNAMAXCUTCAR"
} dbAntennaDataType;
```

Define dbUInt8 for area storage

```
typedef unsigned long long    dbUInt8;
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

The following functions are used for value conversion between `dbUInt8` and `double`.
`dbUserCoord` is defined as `double`.

```
typedef double dbUserCoord;
```

dbUUToDBUArea

`dbUInt8`

```
dbUUToDBUArea (
    dbUserCoord    inputArea,
    dbUserCoord    DBUPerUU
);
```

dbUserToDBUArea

`dbUInt8`

```
dbUserToDBUArea (
    dbCellViewId    cellViewId,
    dbUserCoord    inputArea
);
```

dbDBUAreaToUU

`dbUserCoord`

```
dbDBUAreaToUU (
    dbUInt8    area,
    dbUserCoord    DBUPerUU
);
```

Terminal Antenna Rules

dbDeleteAllTerminalAntennaRules

`Boolean`

```
dbDeleteAllTerminalAntennaRules (
    dbTermId    termId,
    dbAntennaModelType    modelEnum
);
```

Deletes all antenna rules from the specified terminal.

dbHasTerminalAntennaRules

Boolean

```
dbHasTerminalAntennaRules (
    dbTermId          termId,
    dbAntennaModelType modelEnum
);
```

Checks if the terminal has antenna rules.

dbSetTerminalAntennaRules

Boolean

```
dbSetTerminalAntennaRules (
    dbTermId          termId,
    dbAntennaModelType modelEnum,
    dbAntennaDataType dataEnum,
    dbPropType        listType,
    dbPropValue        listValue
);
```

Sets an antenna rule to a terminal according to model type and data type.

dbDeleteTerminalAntennaRules

Boolean

```
dbDeleteTerminalAntennaRules (
    dbTermId          termId,
    dbAntennaModelType modelEnum,
    dbAntennaDataType dataEnum,
    dbPropType        listType,
    dbPropValue        listValue
);
```

Sets an antenna rule to a terminal according to model type and data type.

dbGetTerminalAntennaRules

Boolean

```
dbGetTerminalAntennaRules (
    dbTermId          termId,
    dbAntennaModelType modelEnum,
    dbAntennaDataType dataEnum,
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbPropType          *pListType,  
dbPropValue         *pListValue  
);
```

Returns the terminal antenna rules.

dbStartGenTerminalAntennaRule

```
dbGenStateId  
  
dbStartGenTerminalAntennaRule(  
    dbTermId          termId,  
    dbAntennaModelType modelEnum,  
    dbAntennaDataType dataEnum  
);
```

Starts the generator for antenna rules attached to this terminal, if any.

dbGenTerminalAntennaRule

```
Boolean  
  
dbGenTerminalAntennaRule(  
    dbGenStateId      state,  
    dbPropType        *pType,  
    dbPropValue       *pValue,  
    dbLayer           *layerNum  
);
```

Generates antenna rules attached to this terminal, if any.

dbSetTerminalAntennaRuleOnLayer

```
Boolean  
  
dbSetTerminalAntennaRuleOnLayer(  
    dbTermId          termId,  
    dbAntennaModelType modelEnum,  
    dbAntennaDataType dataEnum,  
    dbLayer           layerNum,  
    dbPropType        type,  
    dbPropValue       value  
);
```

Sets an antenna rule to a specified terminal according to model name, data name, and layer.

dbGetTerminalAntennaRuleOnLayer

Boolean

```
dbGetTerminalAntennaRuleOnLayer(  
    dbTermId          termId,  
    dbAntennaModelType modelEnum,  
    dbAntennaDataType dataEnum,  
    dbLayer           layerNum,  
    dbPropType        *pType,  
    dbPropValue        *pValue  
);
```

Returns the antenna rule for the specified terminal according to model name, data name, and layer.

dbDeleteTerminalAntennaRuleOnLayer

Boolean

```
dbDeleteTerminalAntennaRuleOnLayer(  
    dbTermId          termId,  
    dbAntennaModelType modelEnum,  
    dbAntennaDataType dataEnum,  
    dbLayer           layerNum  
);
```

Deletes the antenna rule of the terminal according to the specified antenna model name, antenna data name, and layer.

Instance Terminals

An instance terminal is a database object that represents a terminal in the master of an instance. Instance terminals, along with instances and nets, are the mechanisms that implement hierarchy. An instance terminal is represented in a name by the `instTerm` part.

Connections for Instance Terminals

An instance terminal can be either connected or unconnected to a net. A connection is always to instance terminals. A connection is represented in a name by the `conn` part.

An instance terminal has one terminal and either one instance or one mosaic instance. A connected instance terminal must have one net. Nets and instances can have many or no connected instance terminals. An instance can have as many instance terminals as there are terminals on the master of the instance.

Instance terminals can be connected in a different order from the order of the terminals on the master of the instance. An instance terminal can have as many instance pins as there are pins on the terminal in the master of the instance.

A member instance terminal, represented in a name by the `memInstTerm` part, refers to a specific member of an instance terminal. A member connection refers to a specific member of a connected instance terminal. The member connection is represented in a name by the `memConn` part of the name. When used in a function name, the member connection implies the presence of an instance terminal ID (`instTermId`) and an integer index. This integer index specifies which bit of the instance terminal, from the left-most bit, is under consideration.

The order of connections between the member nets and the member terminals is significant and is preserved. As the index of a member connection increases, the index for the corresponding member terminal increases until it reaches the number of bits in the terminal. At that point, the index for the member terminal returns to zero, and the index for the corresponding member instance increases by one. This process continues until all bits of the connection have been enumerated.

Note: The number of bits in the connection can either be equal to or not equal to ($=$ or $\neq$) the number of member instances multiplied by the number of member terminals of the instance.

Functions Acting on Instance Terminals

The following functions act on instance terminals. The `Conn` part of the name used in these functions means “connected instance terminal.”

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateInstTerm

```
dbInstTermId
dbCreateInstTerm(
    dbNetId netId, dbAnyInstId anyInstId, dbTermId termId
);
```

dbCreateConn

```
dbInstTermId
dbCreateConn(
    dbNetId netId, dbAnyInstId anyInstId, dbTermId termId
);
```

dbCreateConnByName

```
dbInstTermId
dbCreateConnByName(
    dbNetId netId, dbAnyInstId anyInstId, String termName
);
```

dbCreateConnByNewName

```
dbInstTermId
dbCreateConnByNewName(
    dbNetId netId, dbAnyInstId anyInstId, String newTermName
);
```

The functions `dbCreateInstTerm`, `dbCreateConn`, `dbCreateConnByName`, and `dbCreateConnByNewName` create a new instance terminal and attach it to the specified cell instance. The instance must already exist and be bound to its master.

In the functions `dbCreateInstTerm`, `dbCreateConn`, and `dbCreateConnByName`, the terminal must already exist. In the function `dbCreateConnByName`, the named terminal can be a new terminal that does not yet exist in the associated master. This situation creates an unbound instance terminal.

In the functions `dbCreateConn`, `dbCreateConnByName`, and `dbCreateConnByNewName`, the net must already exist. The newly created instance terminal is then connected to the specified net. The number of bits in the net can differ from the number in the instance terminal. Instances attached to pins or nets are not allowed to have instance terminals.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

In the functions `dbCreateInstTerm`, `dbCreateConn`, `dbCreateConnByName`, and `dbCreateConnByNewName` the parameters `dbAnyInstId` and `InstId` cannot be mosaic instances. `InstTerms` are not supported in mosaic instances.

Note: On the `OpenAccess` database, `instTerms` can also be connected by position. The functions `dbCreateInstTerm`, `dbCreateConn`, `dbCreateConnByName`, and `dbCreateConnByNewName` return `dbcNullInstTermId` if the instance is connected by position. Applications must have checks in place to handle `instTerms` connected by position.

dbDeleteInstTerm

Boolean

```
dbDeleteInstTerm(  
    dbInstTermId instTermId  
);
```

dbDeleteConn

Boolean

```
dbDeleteConn(  
    dbInstTermId instTermId  
);
```

These two functions delete an instance terminal and its instance pins. The instance terminal and its instance pins are disconnected from their nets before being deleted. In the second function, the instance terminal is deleted only if it is connected to a net.

dbAddInstTermToNet

Boolean

```
dbAddInstTermToNet(  
    dbInstTermId instTermId, dbNetId netId  
);
```

Connects an instance terminal to a specified net if it is not already connected. The number of bits in the net does not have to be the same as the number for the instance terminal.

dbSubInstTermFromNet

Boolean

```
dbSubInstTermFromNet(  
    dbInstTermId instTermId  
);
```

Disconnects an instance terminal from its net, if any. The instance terminal is not deleted.

dbIsInstTermBound

Boolean

```
dbIsInstTermBound(  
    dbInstTermId instTermId  
);
```

Tests if this instance terminal is bound to its master.

dbIsInstTermWidthOK

Boolean

```
dbIsInstTermWidthOK(  
    dbInstTermId instTermId  
);
```

Returns TRUE if the number of bits in the instance terminal is equal to the instance terminal in the net connected to it. It returns TRUE if it is unconnected.

dbGetInstTermNet

dbNetId

```
dbGetInstTermNet(  
    dbInstTermId instTermId  
);
```

Returns the net connected to this instance terminal, if any.

dbGetInstTermAnyInst

dbAnyInstId

```
dbGetInstTermAnyInst(  
    dbInstTermId instTermId  
);
```

Returns the instance to which this instance terminal is attached.

dbGetInstTermTerm

dbTermId

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetInstTermTerm(  
    dbInstTermId instTermId  
);
```

Returns the corresponding master terminal for this instance terminal. If unbound, binding is attempted.

dbGetInstTermNetName

```
String  
dbGetInstTermNetName(  
    dbInstTermId instTermId  
);
```

Returns the name of the net of the instance terminal.

dbGetInstTermInstName

```
String  
dbGetInstTermInstName(  
    dbInstTermId instTermId  
);
```

Returns the name of the instance of the instance terminal.

dbGetInstTermTermName

```
String  
dbGetInstTermTermName(  
    dbInstTermId instTermId  
);
```

Returns the name of the terminal of the instance terminal.

Note: On the OpenAccess database, instTerms can also be connected by position. For an instance connected by position, this function attempts to bind an unbound instTerm to obtain a term name. When the attempt fails, NULL is returned. An attempt to bind the instTerm occurs even when no terminal in the master has an assigned position. On the OpenAccess database, there is no guarantee that an application will always get the valid term name for an instTerm created by position. Applications must have checks in place to handle instTerms connected by position.

dbGetMemInstTermMemInst

Boolean

```
dbGetMemInstTermMemInst(  
    dbInstTermId instTermId, int index, dbAnyInstId pAnyInstId, int pInstIndex  
);
```

Returns the corresponding member instance for this member instance terminal.

dbGetMemInstTermMemTerm

Boolean

```
dbGetMemInstTermMemTerm(  
    dbInstTermId instTermId, int index, dbTermId pTermId, int pTermIndex  
);
```

Returns the corresponding master member terminal for this member instance terminal.

dbGetInstTermNumBit

int

```
dbGetInstTermNumBit(  
    dbInstTermId instTermId  
);
```

Returns the number of bits in this instance terminal. This is equivalent to the number of bits in the master terminal multiplied by the number of instances (*numInst*) represented by the instance to which this instance terminal is attached.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateInstTermByPosition

Boolean

```
dbCreateInstTermByPosition(  
    dbNetId          net,  
    dbAnyInstId      anyInstId,  
    unsigned int     position  
);
```

Description

Creates an instTerm that connects to a terminal. The terminal connects to the net at the specified position in the instance master.

The instTerms associated with the instance must all be connected by name or by position.

Arguments

<i>net</i>	The net to use for the connection. A NULL net is allowed.
<i>anyInstId</i>	The instance to connect to the terminal.
<i>position</i>	The position of the terminal in the instance master to which to connect the instTerm.

Return Values

Boolean	Returns TRUE if the instTerm is created. Returns FALSE otherwise.

dbGetInstTermTermPosition

```
unsigned int  
dbGetInstTermTermPosition(  
    dbInstTermId      instTermId  
);
```

Description

Gets the position of the terminal to which the instTerm is bound.

Arguments

<i>instTermId</i>	The instTerm for which to get the terminal position.

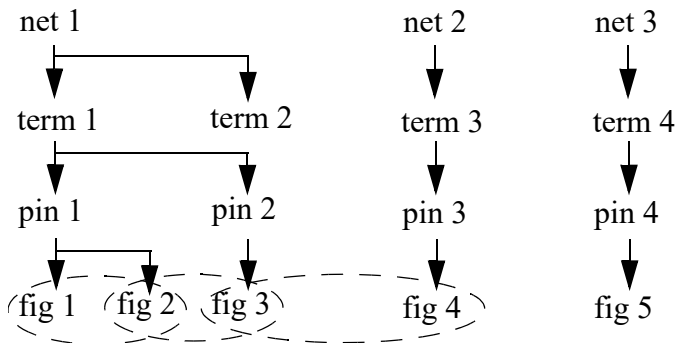
Return Values

<i>unsigned int</i>	Returns the position of the terminal. Returns dbcNullIndex if the instTerm is connected by name or by reference, or if the instTerm cannot be bound to a terminal.

Pins

This section describes functions affecting pin objects. Creating a pin requires a net and a pin name. A figure and a terminal are optional.

The figure below illustrates the OpenAccess pin model.



In the model, term 1 and term 3 know to connect to each other. The model illustrates the following connections.

- Figures on the same pin are *strongly connected*, such as fig 1 and fig 2.
- Figures on different pins on the same terminal are *weakly connected*, such as fig 2 and fig 3.
- Figures on terminals that are on different nets, but the terminals know to connect to each other are *must connect*, such as fig 3 and fig 4. Set the `must join` attribute on terminals to indicate a must connect relationship.
- Fig 5 does not connect to any other figure because term 4 does not *must connect* to any other terminal.
- Terminals on the same net are *shorted* terminals. Figures on shorted terminals are *shorted* to figures on other terminals of the same net. Shorted figures are not strongly or weakly connected. An application interprets the meaning of the shorted terminal.
- Pins are created without figures. You can add 0 to N figures once a pin exists. You can also remove figures from a pin. You can add vias, instances, and shapes to a pin. Via, instance, and shape objects are part of the OpenAccess abstract class `oaPinFig`. You cannot add blockages, boundaries, markers, rows, guides, steiners, or routes to a pin.

dbCreatePin

```
dbPinId  
dbCreatPin(  
    dbNetId      netId ,  
    dbFigId      figId,  
    const char   *pinName  
);
```

Description

Creates a pin.

The function searches the net for terminals. If the search finds more than one terminal, an error results, as the function does not know on which terminal to create the pin. In this case, use `dbCreatePinOnTerm` and specify the terminal Id.

The figure gets a trigger with the *dbcConnModifyAction* type.

Arguments

<i>netId</i>	The net.
<i>figId</i>	The figure. The figure must be part of the <code>oaPinFig</code> class.
<i>*pinNameId</i>	The pin name.

Return Values

<code>dbPinId</code>	Returns the ID of the newly created pin.

dbCreatePinOnTerm

```
dbPinId  
dbCreatPinOnTerm(  
    dbNetId      netId ,  
    dbFigId      figId,  
    const char   *pinName,  
    dbTermId     termId  
);
```

Description

Creates a pin on the specified terminal.

The figure gets a trigger with the *dbcConnModifyAction* type.

Arguments

<i>netId</i>	The net.
<i>figId</i>	The figure. The figure must be part of the oaPinFig class.
<i>*pinNameId</i>	The pin name.
<i>termId</i>	The terminal.

Return Values

dbPinId	Returns the ID of the newly created pin.

dbDeletePin

```
Boolean  
dbDeletePin(  
    dbPinId pinId  
);
```

Deletes this pin. The figure of the pin is detached from the pin, not deleted.

dbGetPinName

```
String
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbGetPinName (
    dbPinId pinId
);
```

Returns the name of the pin.

dbSetPinName

Boolean

```
dbSetPinName (
    dbPinId pinId, String pinName)
```

Sets the pin name. The new pin name must be unique among the pins of the same terminal. If NULL is specified for the `pinName`, a unique name is generated and assigned to this pin.

dbGetPinTerm

dbTermId

```
dbGetPinTerm (
    dbPinId pinId
);
```

Returns the terminal containing this pin.

dbGetPinNet

dbNetId

```
dbGetPinNet (
    dbPinId pinId
);
```

Returns the net to which this pin connects internally. Pins and terminals can point to different nets, but they must be related as parent and child. This relationship can occur with hierarchical nets.

dbMovePinToNet

Boolean

```
dbMovePinToNet (
    dbPinId pinId, dbNetId netId
);
```

Moves the specified pin to the specified net. If the destination net is not in the same net hierarchy as the net currently connected to the pin (that is, they have a different root net), the

root of the destination net must have a terminal. Corresponding instance pins are not unbound if the pin is moved within the same net hierarchy.

dbGetPinFig

```
dbFigId  
dbGetPinFig(  
    dbPinId pinId  
);
```

Returns the first figure generated on the pin.

dbAddFigToPin

```
Boolean  
dbAddFigToPin(  
    dbPinId      pinId ,  
    dbFigId      figId  
);
```

Description

Add the figure to the pin. If the figure is already part of another pin, the function removes the figure from the original pin and adds the figure to the new pin.

When figures are added or removed from a pin, fig triggers are invoked. Application triggers must handle fig triggers, as it is not possible to differentiate between a pin with multiple figures and a pin with only one figure. The figure gets a trigger with the *dbcConnModifyAction* type.

Arguments

<i>pinId</i>	The pin.
<i>figId</i>	The figure. The figure must be part of the <i>oaPinFig</i> class.

Return Values

Boolean	Returns TRUE if the figure is added successfully. Returns FALSE otherwise.

dbSubFigFromPin

Boolean

```
dbSubFigFromPin(  
    dbPinId      pinId ,  
    dbFigId      figId  
);
```

Description

Removes the figure from the pin.

When figures are added or removed from a pin, fig triggers are invoked. Application triggers must handle fig triggers, as it is not possible to differentiate between a pin with multiple figures and a pin with only one figure. The figure gets a trigger with the *dbcConnModifyAction* type.

Arguments

<i>pinId</i>	The pin.
<i>figId</i>	The figure to remove.

Return Values

Boolean	Returns TRUE if the figure is removed successfully. Returns FALSE if the figure was not on the pin.

dbStartGenPinToFig

```
dbGenStateId  
dbStartGenPinToFig(  
    dbPinId      pinId  
);
```

Description

Starts generating all the figures on the pin one at a time.

Arguments

<i>pinId</i>	The pin for which to generate the figures.

Return Values

<code>dbGenStateId</code>	The generator <code>dbGenStateId</code> used to track the state of fig object generation.

dbGenPinToFig

Boolean

```
dbGenPinToFig(  
    dbGenStateId    state,  
    dbFigId         *figId  
);
```

Description

Generates all the figures on the pin one at a time.

Arguments

<i>state</i>	The generator <code>dbGenStateId</code> that tracks the state of fig object generation.
<i>figId</i>	Returns the figure Id.

Return Values

Boolean	Returns TRUE if a fig object is generated successfully. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenNetToTerm

```
dbGenStateId  
dbStartGenNetToTerm(  
    dbNetId      netId  
);
```

Description

Starts generating all the terminals on the net one at a time.

Arguments

<i>netId</i>	The net.

Return Values

dbGenStateId	The generator <code>dbGenStateId</code> used to track the state of term object generation.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGenNetToTerm

Boolean

```
dbGenNetToTerm(  
    dbGenStateId    state,  
    dbTermId        *termId  
);
```

Description

Generating all the terminals on the net one at a time.

Arguments

<i>state</i>	The generator <code>dbGenStateId</code> that tracks the state of term object generation.
<i>termId</i>	Returns the term Id.

Return Values

Boolean	Returns TRUE if a term object is generated successfully. Returns FALSE otherwise.

dbSetTermMustJoin

Boolean

```
dbSetTermMustJoin(  
    dbTermId      termId,  
    dbTermId      mustJoinTermId  
);
```

Description

Sets the must join attribute for the two termIds. The two terms then must connect to each other.

The must join attribute is transitive. If termA is must join to termB, and termB is must join to termC, then termA is must join to termC also.

Arguments

<i>termId</i>	The terminal.
<i>mustJoinTermId</i>	The terminal to which <i>termId</i> must join.

Return Values

Boolean	Returns TRUE if the must join attribute is set for the two terminals. Returns FALSE otherwise.

dbUnsetTermMustJoin

Boolean

```
dbUnsetTermMustJoin(  
    dbTermId    termId  
);
```

Description

Removes the terminal from the must join terminal set.

Arguments

<i>termId</i>	The terminal to removed from the must join terminal set.

Return Values

Boolean	Returns TRUE if the must join attribute is removed from the terminal. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbStartGenTermToMustJoinTerm

```
dbGenStateId  
dbStartGenTermToMustJoinTerm(  
    dbTermId    termId  
);
```

Description

Starts returning all the terminals that are in the must join set of the specified terminal.

Arguments

<i>termId</i>	The terminal.

Return Values

dbGenStateId	The generator <code>dbGenStateId</code> used to track the state of returning the must join terms.

dbGenTermToMustJoinTerm

Boolean

```
dbGenTermToMustJoinTerm(  
    dbGenStateId state,  
    dbTermId *mustJoinTermId  
);
```

Description

Returns all the terminals that are in the must join set of the specified terminal.

Arguments

<i>state</i>	The generator <code>dbGenStateId</code> used to track the state of returning the must join terms.
<i>*mustJoinTermId</i>	A term Id of each terminal in the must join set.

Return Values

Boolean	Returns TRUE if the term object in the must join set is returned. Returns FALSE otherwise.

dbTermHasMustConnectAllPins

```
Boolean  
    dbTermHasMustConnectAllPins(  
        const dbTermId termId  
    );
```

Description

Checks whether the `mustConnectAllPins` attribute is set on the specified terminal.

Note: The `mustConnectAllPins` attribute value can be set only on a single-bit terminal.

Arguments

<i>termId</i>	Database ID of a single-bit terminal.

Return Values

<i>Boolean</i>	Returns TRUE if <code>mustConnectAllPins</code> attribute value is set on the specified terminal.
	Returns FALSE if <code>mustConnectAllPins</code> attribute value is not set or a multi-bit terminal is specified.

dbSetTermMustConnectAllPins

```
void
    dbSetTermMustConnectAllPins (
        dbTermId termId,
        Boolean mustConnectAllPins
    );
```

Description

Indicates whether to set the `mustConnectAllPins` attribute on the specified terminal. The terminal must be a single-bit terminal.

Arguments

<i>termId</i>	Database ID of a single-bit terminal.
<i>mustConnectAllPins</i>	Specifies whether the <code>mustConnectAllPins</code> attribute should be set on the specified terminal. Valid values: ■ TRUE, if the <code>mustConnectAllPins</code> attribute should be set. ■ FALSE, if the <code>mustConnectAllPins</code> attribute should be not set.

Return Value

No value returned.

Access Directions of a Pin

Access directions can be associated with a pin. The access direction is represented by a four-bit wide bit-field. The values are shown in the following table.

Access Directions of a Pin (dbDirection)

Enumerated constant	Description for type dbDirection
<code>dbbNone</code>	The pin is inaccessible.
<code>dbbTop</code>	The pin is accessible from the top.
<code>dbbBottom</code>	The pin is accessible from the bottom.
<code>dbbLeft</code>	The pin is accessible from the left.
<code>dbbRight</code>	The pin is accessible from the right.

When a pin is created, it has no access directions specified. Typically the Abstract Generator (`abgen`) program or the data translators (used by Place and Route) specify pin access directions. A pin can have more than one access direction represented as the logical OR of the bits. For example, a pin that is accessible from both the top and bottom has an access direction of `(dbbTop | dbbBottom)`.

An inaccessible pin is a pin that has an access direction specified, but the access direction has a value of `dbbNone`. The prefix `dbb` identifies a field constant, which is similar to a `dbc` constant except these values can be logically “ORed” together.

dbGetPinAccessDirection

Boolean

```
dbGetPinAccessDirection(  
    dbPinId pinId, dbDirection pDir  
);
```

Returns FALSE if the pin has no access direction specified. Otherwise, the function returns TRUE and `Dir` is the logical OR of the access directions set for the pin.

Note: When the pin access direction value is 0, the function returns TRUE.

dbPinHasAccessDirection

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbPinHasAccessDirection(  
    dbPinId pinId  
);
```

Returns TRUE if pin access directions are specified.

dbSetPinAccessDirection

Boolean

```
dbSetPinAccessDirection(  
    dbPinId pinId, dbDirection dir  
);
```

Sets access directions for the pin. If an access direction of `dbbNone` is specified, then the pin is inaccessible. For `dbDirection` values, see [“Pins”](#) on page 748.

dbDeletePinAccessDirection

Boolean

```
dbDeletePinAccessDirection(  
    dbPinId pinId  
);
```

Removes any specifications for access directions for the pin.

dbTransformAccessDirection

dbDirection

```
dbTransformAccessDirection(  
    dbDirection direction, dbOrient orient  
);
```

Transforms the access direction based on the specified orientation.

BusDef and BitOrder

This section describes the functions affecting bus definition and bit order objects.

There are three bus definition classes, `oaBusNetDef`, `oaBusTermDef`, and `oaVectorInstDef`. These three bus definition objects define a vectored object such as `busNet`, `busTerm`, or `vectorInst`. The definition includes base name, bit range, and bit order. The same bus definition manages all vectored objects in a cellview with the same base name.

The bus definition class tracks the minimum and maximum indexes referred to by all the corresponding bus and `busBit` objects. A bus definition can have missing bits. Buses need not start or end at zero, but the indexes must be greater than or equal to zero.

A bus definition object can be implicit or explicit.

If a bus definition object does not already exist when a bus object is created, the database automatically creates an implicit bus definition object. The database adds vectored objects with the same base name to the vector definition.

You can explicitly create a bus definition to specify a bit order. The database does not automatically destroy explicit bus definitions even after the last vectored object with the same base name is destroyed. The database does automatically destroy implicit bus definitions when the last vectored object with the same base name is destroyed. To destroy an explicit bus definition, applications must destroy it explicitly.

The three bus definition classes, `oaBusNetDef`, `oaBusTermDef`, and `oaVectorInstDef` support a `bitOrder` attribute. You can set a `bitOrder` attribute when creating a bus definition object or by using the `setBitOrder` method.

The database implicitly creates vector definitions, if they do not exist. Implicit vector definitions have a `bitOrder` value of `none`.

Three ids support the functions.

```
typedef dbId dbBusNetDefId;  
typedef dbId dbBusTermDefId;  
typedef dbId dbVectorInstDefId;
```

The following validation functions check whether the database bus definition Id maps to a valid `OpenAccess` bus definition object.

```
inline oaBusNetDef *  
dbiValidateBusNetDef (dbBusNetDefId id);  
  
inline oaBusTermDef *
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
dbiValidateBusTermDef (dbBusTermDefId id);  
  
inline oaVectorInstDef *  
dbiValidateVectorInstDef (dbVectorInstDefId id);
```

The following enumeration type supports bitOrder.

```
typedef enum dbBitOrder {  
    dbcNoBitOrder          = 0,  
    dbcAscendingBitOrder    = 1,  
    dbcDescendingBitOrder   = 2  
} dbBitOrder;
```

dbCreateBusNetDef

```
dbBusNetDefId  
dbCreateBusNetDef(  
    dbCellViewId      cvId,  
    const char*       baseName,  
    dbBitOrder         order  
);
```

Description

Creates a busNet definition in the cellview using the bitOrder and baseName, if a busNet definition does not already exist.

Arguments

<i>cvId</i>	Cellview in which to create the busNetDef.
<i>baseName</i>	Base name for busNetDef.
<i>order</i>	Bit order for the busNetDef.

Return Values

<i>dbBusNetDefId</i>	Returns the ID of the created busNetDef. If an oaBusNetDef already exists, returns a NULL ID and issues a warning.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbCreateBusTermDef

```
dbBusTermDefId  
dbCreateBusTermDef(  
    dbCellViewId          cvId,  
    const char*           baseName,  
    dbBitOrder             order  
);
```

Description

Creates a busTerm definition in the cellview using the bitOrder and baseName, if a busTerm definition does not already exist.

Arguments

<i>cvId</i>	Cellview in which to create the busTermDef.
<i>baseName</i>	Base name for busTermDef.
<i>order</i>	Bit order for the busTermDef.

Return Values

dbBusTermDefId	Returns the ID of the created busTermDef. If an oaBusTermDef already exists, returns a NULL ID and issues a warning.

dbCreateVectorInstDef

```
dbVectorInstDefId  
dbCreateVectorInstDef(  
    dbCellViewId          cvId,  
    const char*           baseName,  
    dbBitOrder             order  
);
```

Description

Creates a vectorInst definition in the cellview using the bitOrder and baseName, if a vectorInst definition does not already exist.

Arguments

<i>cvId</i>	Cellview in which to create the vectorInstDef.
<i>baseName</i>	Base name for vectorInstDef.
<i>order</i>	Bit order for the vectorInstDef.

Return Values

<i>dbVectorInstDefId</i>	Returns the ID of the created vectorInstDef. If an oavectorInstDef already exists, returns a NULL ID and issues a warning.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindBusNetDef

```
dbBusNetDefId  
dbFindBusNetDef(  
    dbCellViewId      cvId,  
    const char*        baseName  
);
```

Description

Searches the cellview for an oaBusNetDef with the specified baseName.

Arguments

<i>cvId</i>	Cellview in which to search.
<i>baseName</i>	Base name for which to search.

Return Values

dbBusNetDefId	Returns the ID of the busNetDef. If the busNetDef does not exist, returns a NULL.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindBusTermDef

```
dbBusTermDefId  
dbFindBusTermDef(  
    dbCellViewId      cvId,  
    const char*        baseName  
);
```

Description

Searches the cellview for an oaBusTermDef with the specified baseName.

Arguments

<i>cvId</i>	Cellview in which to search.
<i>baseName</i>	Base name for which to search.

Return Values

dbBusTermDefId	Returns the ID of the busTermDef. If the busTermDef does not exist, returns a NULL.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindVectorInstDef

```
dbVectorInstDefId
dbFindVectorInstDef(
    dbCellViewId      cvId,
    const char*        baseName
);
```

Description

Searches the cellview for an oaVectorInstDef with the specified baseName.

Arguments

<i>cvId</i>	Cellview in which to search.
<i>baseName</i>	Base name for which to search.

Return Values

dbVectorInstDefId	Returns the ID of the vectorInstDef. If the vectorInstDef does not exist, returns a NULL.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbDeleteBusNetDef

Boolean

```
dbDeleteBusNetDef(  
    dbBusNetDefId      busNetDefId  
);
```

Description

Deletes the busNetDef from the cellview.

Arguments

<i>busNetDefId</i>	The busNetDef to delete.

Return Values

Boolean	Returns TRUE if the busNetDef is deleted. Returns FALSE otherwise.

dbDeleteBusTermDef

Boolean

```
dbDeleteBusTermDef(  
    dbBusTermDefId      busTermDefId  
);
```

Description

Deletes the busTermDef from the cellview.

Arguments

<i>busTermDefId</i>	The busTermDef to delete.

Return Values

Boolean	Returns TRUE if the busTermDef is deleted. Returns FALSE otherwise.

dbDeleteVectorInstDef

Boolean

```
dbDeleteVectorInstDef(  
    dbVectorInstDefId      vectorInstDefId  
);
```

Description

Deletes the vectorInstDef from the cellview.

Arguments

<i>vectorInstDefId</i>	The vectorInstDef to delete.

Return Values

Boolean	Returns TRUE if the vectorInstDef is deleted. Returns FALSE otherwise.

Attribute Access Functions

dbGetBusNetDefMinIndex

```
int  
dbGetBusNetDefMinIndex (  
    dbBusNetDefId          busNetDefId  
);
```

Description

Gets the smallest bit number used for the busNetDef object. This is the smallest index used across all busNets associated with this busNetDef object.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.

Return Values

<i>int</i>	Returns the smallest bit number used for the busNetDef object.

dbGetBusNetDefMaxIndex

```
int  
dbGetBusNetDefMaxIndex (  
    dbBusNetDefId          busNetDefId  
);
```

Description

Gets the largest bit number used for the busNetDef object. This is the largest index used across all busNets associated with this busNetDef object.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.

Return Values

int	Returns the largest bit number used for the busNetDef object.

dbGetBusNetDefBaseName

String

```
dbGetBusNetDefBaseName (  
    dbBusNetDefId          busNetDefId  
);
```

Description

Gets the base name of the busNetDef object.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.

Return Values

String	Returns the base name of the busNetDef object.

dbGetBusNetDefNumBits

```
int  
dbGetBusNetDefNumBits (  
    dbBusNetDefId          busNetDefId  
);
```

Description

Gets the number of bits covered by the busses in this busNetDef object. The number is equal to $\text{abs}(\text{maxIndex} - \text{minIndex}) + 1$.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.

Return Values

int	Returns the number of bits covered by the busses in this busNetDef object.

dbGetBusNetDefNumBitsUsed

```
int  
dbGetBusNetDefNumBitsUsed(  
    dbBusNetDefId      busNetDefId  
);
```

Description

Gets the number of bits used by the busses in this busNetDef object. The number is equal to the number of busNetBits in this busNetDef.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.

Return Values

int	Returns the number of bits used by the busses in this busNetDef object.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbGetBusNetDefBitOrder

```
dbBitOrder  
dbGetBusNetDefBitOrder(  
    dbBusNetDefId      busNetDefId  
);
```

Description

Gets the bitOrder of the busNetDef object.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.

Return Values

dbBitOrder	Returns the bitOrder of the busNetDef object.

dbSetBusNetDefBitOrder

Boolean

```
dbSetBusNetDefBitOrder(  
    dbBusNetDefId      busNetDefId,  
    dbBitOrder          order  
);
```

Description

Sets the bitOrder on the busNetDef object. The busNetDef must be explicit.

Arguments

<i>busNetDefId</i>	The ID of the busNetDef object.
<i>order</i>	The new bitOrder.

Return Values

Boolean	Returns TRUE if the bitOrder is set. Returns FALSE if the busNetDef is implicit.

dbGetBusTermDefMinIndex

```
int  
dbGetBusTermDefMinIndex(  
    dbBusTermDefId      busTermDefId  
);
```

Description

Gets the smallest bit number used for the busTermDef object. This is the smallest index used across all busTerms associated with this busTermDef object.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.

Return Values

<i>int</i>	Returns the smallest bit number used for the busTermDef object.

dbGetBusTermDefMaxIndex

```
int  
dbGetBusTermDefMaxIndex (  
    dbBusTermDefId      busTermDefId  
);
```

Description

Gets the largest bit number used for the busTermDef object. This is the largest index used across all busTerms associated with this busTermDef object.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.

Return Values

<i>int</i>	Returns the largest bit number used for the busTermDef object.

dbGetBusTermDefBaseName

String

```
dbGetBusTermDefBaseName (  
    dbBusTermDefId      busTermDefId  
);
```

Description

Gets the base name of the busTermDef object.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.

Return Values

String	Returns the base name of the busTermDef object.

dbGetBusTermDefNumBits

```
int  
dbGetBusTermDefNumBits (  
    dbBusTermDefId      busTermDefId  
);
```

Description

Gets the number of bits covered by the busses in this busTermDef object. The number is equal to $\text{abs}(\text{maxIndex} - \text{minIndex}) + 1$.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.

Return Values

int	Returns the number of bits covered by the busses in this busTermDef object.

dbGetBusTermDefNumBitsUsed

```
int  
dbGetBusTermDefNumBitsUsed(  
    dbBusTermDefId      busTermDefId  
);
```

Description

Gets the number of bits used by the busses in this busTermDef object. The number is equal to the number of busTermBits.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.

Return Values

<i>int</i>	Returns the number of bits used by the busses in this busTermDef object.

dbGetBusTermDefBitOrder

```
bitOrder  
dbGetBusTermDefBitOrder(  
    dbBusTermDefId      busTermDefId  
);
```

Description

Gets the bitOrder of the busTermDef.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.

Return Values

<i>bitOrder</i>	Returns the bitOrder of the busTermDef.

dbSetBusTermDefBitOrder

Boolean

```
dbSetBusTermDefBitOrder(  
    dbBusTermDefId      busTermDefId,  
    dbBitOrder           order  
);
```

Description

Sets the bitOrder of the busTermDef.

Arguments

<i>busTermDefId</i>	The ID of the busTermDef object.
<i>order</i>	The new bitOrder for the busTermDef.

Return Values

Boolean	Returns TRUE if the bitOrder is set. Returns FALSE if the busTermDef is implicit.

dbGetVectorInstDefMinIndex

```
int  
dbGetVectorInstDefMinIndex(  
    dbVectorInstDefId      vectorInstDefId  
);
```

Description

Gets the smallest bit number used for the vectorInst object. This is the smallest index used across all vectorInsts associated with the busNetDef object.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.

Return Values

<i>int</i>	Returns the smallest bit number used for the vectorInst object.

dbGetVectorInstDefMaxIndex

```
int  
dbGetVectorInstDefMaxIndex(  
    dbVectorInstDefId      vectorInstDefId  
);
```

Description

Gets the largest bit number used for the vectorInst object. This is the largest index used across all vectorInsts associated with the busNetDef object.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.

Return Values

<i>int</i>	Returns the largest bit number used for the vectorInst object.

dbGetVectorInstDefBaseName

String

```
dbGetVectorInstDefBaseName (
    dbVectorInstDefId      vectorInstDefId
);
```

Description

Gets the base name of the vectorInst object.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.

Return Values

String	Returns the base name of the vectorInst object.

dbGetVectorInstDefNumBits

```
int  
dbGetVectorInstDefNumBits(  
    dbVectorInstDefId      vectorInstDefId  
);
```

Description

Gets the number of bits covered by the busses in the vectorInstDef object. The number is equal to $\text{abs}(\text{maxIndex} - \text{minIndex}) + 1$.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.

Return Values

<i>int</i>	Returns the number of bits covered by the busses in the vectorInstDef object.

dbGetVectorInstDefNumBitsUsed

```
int  
dbGetVectorInstDefNumBitsUsed(  
    dbVectorInstDefId      vectorInstDefId  
);
```

Description

Gets the number of bits used by the busses in the vectorInstDef object. The number is equal to the number of vectorInstBits.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.

Return Values

<i>int</i>	Returns the number of bits used by the busses in the vectorInstDef object.

dbGetVectorInstDefBitOrder

```
dbBitOrder  
dbGetVectorInstDefBitOrder(  
    dbVectorInstDefId      vectorInstDefId  
);
```

Description

Gets the bitOrder for the vectorInstDef.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.

Return Values

<i>dbBitOrder</i>	Returns the vectorInstDef bitOrder.

dbSetVectorInstDefBitOrder

Boolean

```
dbSetVectorInstDefBitOrder(  
    dbVectorInstDefId      vectorInstDefId,  
    dbBitOrder              order  
);
```

Description

Sets the bitOrder for the vectorInstDef.

Arguments

<i>vectorInstDefId</i>	The ID of the vectorInstDef object.
<i>order</i>	The bit order.

Return Values

Boolean	Returns TRUE if the bitOrder is set. Returns FALSE if the busTermDef is implicit.

Cellview Information

The following functions perform operations and retrieve information on objects found in cellviews.

dbCellViewHasOpticalData

Boolean

```
dbCellViewHasOpticalData(  
    dbCellViewId cvId  
);
```

Description

Checks whether the specified cellview has optical data.

Arguments

<i>dbCellViewId</i>	The database ID of a cellview ID.

Return value

<i>Boolean</i>	TRUE, if the cellview has optical data.
	FALSE, if the cellview does not contain optical data.

Example

```
dbCellViewHasOpticalData(cv_id)
```

dbDeleteAllNet

Boolean

```
dbDeleteAllNet(  
    dbCellViewId cellViewId)
```

Deletes all nets in a cellview.

dbFindNetByName

```
dbNetId  
dbFindNetByName (  
    dbCellViewId cellViewId, String name  
);
```

Finds a net in a cellview by name. The net must be a top-level net.

dbFindSigByName

```
dbSigId  
dbFindSigByName (  
    dbCellViewId cellViewId, String name  
);
```

Finds a signal in a cellview by name.

dbFindSigByAlias

```
dbSigId  
dbFindSigByAlias (  
    dbCellViewId cellViewId, String alias  
);
```

Finds a signal in a cellview by its alias.

dbFindAnyInstByName

```
dbAnyInstId  
dbFindAnyInstByName (  
    dbCellViewId cellViewId, String name)
```

Finds any instance in a cellview by name.

dbFindMemInstByName

```
Boolean  
dbFindMemInstByName (  
    dbCellViewId cellViewId, String memName, dbAnyInstId pAnyInstId, unsigned int  
    pInstIndex  
);
```

Finds a member instance in a cellview by member name.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

dbFindTermByName

```
dbTermId  
dbFindTermByName(  
    dbCellViewId cellViewId, String name)
```

Finds a terminal in a cellview by name.

dbHasTerm

```
Boolean  
dbHasTerm(  
    dbCellViewId cellViewId  
);
```

Returns TRUE if the specified cellview has at least one terminal. Otherwise, the function returns FALSE.

dbStartGenTerm

```
dbGenStateId  
dbStartGenTerm(  
    dbCellViewId cellViewId  
);
```

Starts the generator for terminals of the cellview.

dbGenTerm

```
Boolean  
dbGenTerm(  
    dbGenStateId state, dbTermId pTermId  
);
```

Generates terminals of the cellview.

dbStartGenMemTermByName

```
dbGenStateId  
dbStartGenMemTermByName(  
    dbCellViewId cellViewId, String memName  
);
```


dbGenMemTermByName

Boolean

```
dbGenMemTermByName (
    dbGenStateId state, dbTermId pTermId, unsigned int pTermIndex
);
```

Generates terminals containing the specified member name.

dbStartGenNet

dbGenStateId

```
dbStartGenNet (
    dbCellViewId cellViewId
);
```

Starts the generator for top-level nets.

dbGenNet

Boolean

```
dbGenNet (
    dbGenStateId state, dbNetId pNetId
);
```

Generates top-level nets.

dbStartGenMemInst

dbGenStateId

```
dbStartGenMemInst (
    dbCellViewId cellViewId
);
```

Starts the generator for member instances.

dbGenMemInst

Boolean

```
dbGenMemInst (
    dbGenStateId state, dbAnyInstId pAnyInstId, int pInstIndex
);
```

Generates member instances.

dbStartGenSig

```
dbGenStateId  
dbStartGenSig(  
    dbCellViewId cellViewId  
);
```

Starts the generator for unique signals in this cellview.

dbGenSig

```
Boolean  
dbGenSig(  
    dbGenStateId state, dbSigId pSigId  
);
```

Generates unique signals in this cellview.

dbStartGenConn

```
dbGenStateId  
dbStartGenConn(  
    dbCellViewId cellViewId  
);
```

Starts the generator for all connected instance terminals contained in this cellview.

dbGenConn

```
Boolean  
dbGenConn(  
    dbGenStateId state, dbInstTermId pInstTermId  
);
```

Generates all connections (all connected instance terminals) contained in this cellview.

Equivalent Connectivity Time Functions

The following functions perform operations like setting, retrieving, and removing equivalent connectivity time on a cellview.

dbCellViewHasEquivalentConnectivityTime

Boolean

```
dbCellViewHasEquivalentConnectivityTime(  
    dbCellViewId cvId  
)
```

Description

Checks whether the equivalent connectivity time is set on the specified cellview.

Arguments

<i>cvId</i>	Database ID of a cellview.

Return Value

Boolean	TRUE, if a valid equivalent connectivity time is set on the specified cellview. FALSE, if no valid equivalent connectivity time is set on the specified cellview.

Example

```
dbCellViewHasEquivalentConnectivityTime(cvId) => true
```

Confirms that a valid equivalent connectivity time is set on the cellview *cvId*.

dbGetCellViewEquivalentConnectivityTime

Boolean

```
dbGetCellViewEquivalentConnectivityTime  
    dbCellViewId cvId  
    time_t *equivalentTime  
    )
```

Description

Retrieves the equivalent connectivity time that is set on the specified cellview.

Arguments

<i>cvId</i>	Database ID of a cellview
<i>*equivalentTime</i>	Pointer for a time type to store the retrieved equivalent time.

Return Value

<i>Boolean</i>	TRUE, if the equivalent connectivity time is retrieved from the specified cellview. FALSE, if the equivalent connectivity time is not retrieved from the specified cellview. Equivalent time will be set to zero if equivalent connectivity time is not set.

Example

```
dbGetCellViewEquivalentConnectivityTime(cvId, equivalentTime) => true
```

Successfully retrieved the equivalent connectivity time that is set on the specified cellview.

dbSetCellViewEquivalentConnectivityTime

Boolean

```
dbSetCellViewEquivalentConnectivityTime  
    dbCellViewId cvId  
    timeString time_t  
    )
```

Description

Sets the equivalent connectivity time on the specified cellview.

Arguments

<i>cvId</i>	Database ID of a cellview.
<i>time_t</i>	Equivalent connectivity time to be set on the cellview.

Return Value

<i>Boolean</i>	TRUE, if equivalent connectivity time is set on the cellview. FALSE, if equivalent connectivity time is not set.

Example

```
dbSetCellViewEquivalentConnectivityTime(cvId, currentTime) => true
```

Successfully retrieved the equivalent connectivity time that is set on the specified cellview.

dbUnsetCellViewEquivalentConnectivityTime

Boolean

```
dbUnsetCellViewEquivalentConnectivityTime (  
    dbCellViewId cvId  
)
```

Description

Removes equivalent connectivity time information from the specified cellview

Arguments

<i>cvId</i>	Database ID of a cellview.

Return Value

<i>Boolean</i>	TRUE, if equivalent connectivity time information is removed from the specified cellview. FALSE, if the operation fails.

Example

```
dbUnsetCellViewEquivalentConnectivityTime(cvId) => true
```

Equivalent connectivity time information is removed from the cellview *cvId*. Name Information

The following functions retrieve information about or act on names of nets, terminals, or instances.

dbGetMemName

String

```
dbGetMemName (  
    String name, int memIndex  
);
```

Retrieves the name of the indexed member from the specified name and returns it as a permanent string. For example, given a name (A<15 : 0>, CLK) and a member index 2, this function returns the member A<13>.

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

In case of an error, the specified member name returned is composed of the specified `name` suffixed by the specified `<memIndex>`. For example, if a member index is 17, a warning message is produced and the string returned is `(A<15:0>,CLK)<17>`.

dbGetNameNumBit

```
int
dbGetNameNumBit(
    String name
);
```

Evaluates the specified name and returns the number of bits implied by the name. The function returns zero if the name specified is not expressed in legal syntax.

The function `dbGetInstNameNumInst` is documented in [Chapter 7, “Instance and Hierarchy Functions.”](#)

dbProduceMemName

```
Boolean
dbProduceMemName(
    String name, FuncBooleanPtr myMemNameConsumer, unsigned char pConsumerData
);
```

Produces the names of members from the specified name. For each member name produced, the consumer function is called with the specified consumer data and a pointer to a static buffer that contains the member name.

The consumer data is a pointer to a data structure that you define. The consumer data can also be a NULL pointer. If the consumer function returns TRUE, the producer continues to produce member names. If the consumer function returns FALSE, the producer stops the member name production and returns TRUE.

myMemNameConsumer

The consumer function for the `dbProduceMemName` function is declared as follows:

```
Boolean
myMemNameConsumer(
    String memName, myData pMyData
);
```

Net and Terminal Name Syntax

The name syntax for both nets and terminals is shown below in BNF format (Backus Naur Format) notation. For more information on BNF format, see *Design Data Translator's Reference*.

Square brackets ([]) indicate that the item enclosed is optional. Either zero or at most one item can be enclosed in brackets. Items enclosed in curly braces ({ }) are also optional, but zero or more items can be enclosed. The vertical bar (|) implies the relationship **or**. The word **number** represents a nonnegative integer less than 65536.

```
nameExpr ::= [ < * number > ] nameTerm { , nameExpr }
nameTerm ::= baseName [ < vectExpr > ] | < nameExpr >
vectExpr ::= vectTerm [ * number ] { , vectExpr }
vectTerm ::= number [ : number [ : number ] ] | < vectExpr >
```

nameExpr

A string consisting of a name term that is optionally prefixed with the **<*N>** construct (not shown in the example syntax). This construct is also referred to as a prefix repeat operator, indicating that the name term is repeated (enumerated) **N** times. The value of **N** must be larger than zero. 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 parentheses (not shown in the example syntax).

baseName

A string of printable ASCII characters that does not include reserved special characters, such as angle brackets (< >), parentheses (()), commas (,), slashes (/) or white space. Parentheses (()) are allowed in a base name only if they are enclosing a number.

vectExpr

A single or repeated vector term. The optional ***N** suffix on a vector term (not shown in the example syntax) is called a suffix repeat operator and 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 by a starting, ending, and number increments separated by colons. For example, the range `<3:0:2>` indicates a starting number of 3 decremented by 2 to member 0, indicating two member numbers 3 and 1 in the expanded form.

A vector term can also be a vector expression enclosed in parentheses (not shown in the example syntax). In this case, the innermost term is expanded first, then the repeat operator is applied to repeat the entire expanded term enclosed in parentheses.

Name and Vector Expression Expansion

The following table gives examples of how name and vector expressions are expanded. The expanded form of a name is semantically equivalent to the original name. You create the expanded form by expanding the range specifications, applying repeat operators, and removing parentheses in the original name.

Use a zero-based counting scheme to find the Nth member in the expanded form. For example, member zero of $\langle *2 \rangle(a, \langle *2 \rangle b)$ is a, member one is b, and member two is b. If the name contains a vector expression, such as $b\langle 0:1, 2:2 \rangle$, then member zero in this case is called $b\langle 0 \rangle$, member one is $b\langle 1 \rangle$, and member two is $b\langle 2 \rangle$.

Name	Number of members	Expanded form
clk	1	clk
data $\langle 2 \rangle$	1	data $\langle 2 \rangle$
$\langle *1 \rangle$ base	1	base
$\langle *2 \rangle$ term	2	term,term
$\langle *2 \rangle(a,b),c$	5	a,b,a,b,c
$\langle *2 \rangle(a, \langle *2 \rangle b)$	6	a,b,b,a,b,b
$b\langle 0:2 \rangle$	3	$b\langle 0,1,2 \rangle$
$b\langle 0:2:1 \rangle$	3	$b\langle 0,1,2 \rangle$
$b\langle 3:0:2 \rangle$	2	$b\langle 3,1 \rangle$
$b\langle 0:2*2 \rangle$	6	$b\langle 0,0,1,1,2,2 \rangle$
$b\langle (0:2)*2 \rangle$	6	$b\langle 0,1,2,0,1,2 \rangle$
$b\langle 0,2*2 \rangle$	3	$b\langle 0,2,2 \rangle$
$b\langle (0,2)*2 \rangle$	4	$b\langle 0,2,0,2 \rangle$
$b\langle 0,1:3:4*1,2:2 \rangle$	3	$b\langle 0,1,2 \rangle$
$b\langle 0:1,2:2 \rangle$	3	$b\langle 0,1,2 \rangle$

Automatic Database Deletions

Some actions occur automatically within the database to maintain consistency of the data. These actions are associated with deleting the following objects:

- If a net is deleted, its terminal and pins are deleted because these objects cannot exist without a net.
- If a net is deleted, its instance pins and instance terminals are disconnected from that net.
- If a pin or a net is deleted, its figures become disconnected from that pin or net but remain as figures in the cellview.
- If an instance is deleted, the instance terminals and instance pins associated with that instance are disconnected from their nets and deleted.
- If a terminal is deleted, the instance terminals that refer to this terminal from other cellviews are unbound, and the pins attached to this terminal are deleted.
- If a pin is deleted, the instance pins that refer to this pin from other cellviews are unbound.
- If a figure belonging to a pin is deleted, the corresponding pin is deleted.

Constraints on Database Objects

The following rules are additional constraints on the usage and relationships between objects in the database. These constraints are imposed in addition to those implied by the structure of the schema presented in [“CDBA Database Objects”](#) on page 109.

- A net cannot have the same name as any other net, unless both or all nets have the same root net (that is, both or all nets are logically carrying the same electrical signals).
- Two nets that have the same root net are logically equivalent. If they have different names, both names are aliases for the net. The name of the root net is the preferred name for the net.
- A signal that has a global scope must have a terminal and a net. This is not currently enforced by the system.
- The system makes connections automatically between signals occurring in different nets according to the names of the signals.

Restrictions to the Interface

The following is a list of restrictions on the interface due to the current physical data model.

- A terminal cannot be created for a net that has a signal already attached to a member terminal with a different name. A signal can be connected to more than one member terminal through its nets as long as those member terminals have the same name.
- The `globalDownOne` scope for nets is not implemented. It is defined in EDIF as `global`.
- Terminals for global signals are not automatically created.
- Net names only support one-dimensional indexing.
- Modifying connectivity while using generators can be hazardous.

Connectivity Example Function

The following example program implements connectivity between an inverter and two terminals. The program uses some of the functions described in this chapter.

```
/*
 * Simple schematic containing one inverter and two
 * terminals.
 */

#include "db.h"

createExample1(dbCellViewId cellViewId)
{
    dbCellViewId      invCellViewId;
    dbInstId          instId;
    dbNetId            netId;
    dbInstTermId       connId;
    dbPoint            origin;

    /*
     * Leave out error status check for clarity.
     * Create an inverter instance and hook it up to nets
     * and terminals.
     */

    invCellViewId = dbOpenCellViewByType("inv", "symbol", "r",
                                         dbcNullCellViewId, "r", NULL);

    origin.x = origin.y = 0;

    instId = dbCreateInst(cellViewId, invCellViewId, "inv1",
                          origin, dbcR0, 1);

    netId = dbCreateNet(cellViewId, "N1", dbcNullNetId);
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

```
termId = dbCreateTerm(netId, "TI", dbcInput, NULL);
dbCreateTerm(netId, "TI", dbcInput, NULL);

connId = dbCreateConnByName(netId, (dbAnyInstId) instId,
                             "A");

netId = dbCreateNet(cellViewId, "N2", dbcNullNetId);

termId = dbCreateTerm(netId, "TO", dbcOutput, NULL);
dbCreateTerm(netId, "TO", dbcOutput, NULL);

connId = dbCreateConnByName(netId, (dbAnyInstId) instId,
                             "Y");
}
```

Cadence Integrators Toolkit Database Reference

CDBA Figure Functions

CDBA Placement Functions

This chapter includes the following topics:

- [Structs in Placement Infrastructure](#)
- [Enumerated Types and Enumerated Constants used in Placement Infrastructure](#)
- [Background Definition Functions](#)
- [Component Type Set Definition Functions](#)
- [Rail Definition Functions](#)
- [Placement Area Definition Functions](#)
- [Row-based Functions](#)
- [Row Region Specification Functions](#)
- [Row Region Functions](#)
- [Miscellaneous Functions](#)

Structs in Placement Infrastructure

This section covers the structs used in placement infrastructure.

dbBackgroundDefAttrData

Type	Name	Description
dbLayer	layer	Layer associated with backGroundDef.
dbPurpose	purpose	Purpose associated with backGroundDef.
String	netName	Name of a net.
int	leftEnc	Left enclosure of the background shape.
int	rightEnc	Right enclosure of the background shape.
int	topEnc	Top enclosure of the background shape.
int	botEnc	Bottom enclosure of the background shape.

dbCompFilterSpec

Type	Name	Description
String	libPattern	Library name pattern.
String	cellPattern	Cell name pattern.
String	viewPattern	View name pattern.

dbCompTypeSetDefAttrData

Type	Name	Description
dbAlignmentType	refOnArea	Alignment type of area reference.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbCompTypeSetDefAttrData

Type	Name	Description
dbAlignmentType	refOnComp	Alignment type of component reference.
dbOrient	*orients	Allowed orientations that placer can use for cells in the component types.
unsigned int	numOrients	Number of allowed orientations.
String	*allowedTypes	Allowed component types that share placement attributes.
unsigned int	numAllowedTypes	Number of component types.
dbCompFilterSpec	*compFilters	Allowed component filters.
unsigned int	numCompFilters	Number of allowed component filters.
int	offset	Offset from the reference points computed from row reference, specified in user units.
int	grid	Pitch or grid to use between placement locations.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbPlaceRowAttrData

Text	Name	Description
String	name	Name of the placeArea.
unsigned int	offset	Offset from the reference points computed from row reference, specified in user units.
dbOrient	orient	Orientation of the placeArea
dbRailDefAttrData	*railDefAttrs	Pointer to array with railDef attributes.
unsigned int	numRailDefAttrs	Pointer to the memory area where the number of railDef attributes is placed.
dbCompTypeSetDefAttrData	*compTypeSetDefAttrs	Pointer to array with compTypeSetDef attributes.
unsigned int	numCompTypeSetDefAttrs	Pointer to the memory area where the number of compTypeSetDef attributes is placed.
dbBackgroundDefAttrData	*backgroundDefAttrs	Pointer to array with backgroundDef attributes.
unsigned int	numBackgroundDefAttrs	Pointer to the memory area where the number of backgroundDef attributes is placed.
dbPlaceRowType	tag	Placement type of row.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbPlaceRowAttrData

Text	Name	Description
<code>union rowInfo</code>		Specify height or siteDef for the placeArea.
<code>unsigned int</code>	<code>height</code>	Height of the placeArea in user units.
<code>String</code>	<code>siteDef</code>	siteDef for placeArea.

dbRailDefAttrData

Text	Name	Description
<code>dbLayer</code>	<code>layer</code>	Layer associated with railDef.
<code>dbPurpose</code>	<code>purpose</code>	Purpose associated with railDef.
<code>dbAlignmentType</code>	<code>refOnArea</code>	Alignment type of area reference.
<code>dbAlignmentType</code>	<code>refOnRail</code>	Alignment type of rail reference.
<code>unsigned int</code>	<code>width</code>	Width of railDef.
<code>String</code>	<code>netName</code>	Name of a net.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbRailDefAttrData

Text	Name	Description
int	offset	Places the rail 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.
int	beginOffset	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.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbRailDefAttrData

Text	Name	Description
int	endOffset	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.
dbRailFigType	figType	figType of the railDef.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbPlaceCellsAttrData

Type	Name	Description
String	leftBoundaryCell	Name of the left boundary cell.
Boolean	leftBoundaryCellMirrored	Specifies whether the left boundary cell is mirrored.
String	rightBoundaryCell	Name of the right boundary cell.
Boolean	rightBoundaryCellMirrored	Specifies whether the right boundary cell is mirrored.
String	topLeftBoundaryCell	Name of the top-left boundary cell.
String	bottomLeftBoundaryCell	Name of the bottom-left boundary cell.
Boolean	leftCornerCellsMirrored	Specifies whether the left corner cell is mirrored.
String	topRightBoundaryCell	Name of the top right boundary cell.
String	bottomRightBoundaryCell	Name of the bottom right boundary cell.
Boolean	rightCornerCellsMirrored	Specifies whether the right corner cell is mirrored.
String	*topBoundaryCells	List of top boundary cells.
unsigned int	numTopBoundaryCells	Number of allowed top boundary cells.
String	*bottomBoundaryCells	List of bottom boundary cells.
unsigned int	numBottomBoundaryCells	Number of allowed bottom boundary cells.
String	*tapCells	List of tap cells.
unsigned int	numTapCells	Number of allowed tap cells.
String	*fillCells	List of fill cells.
unsigned int	numFillCells	Number of allowed fill cells.
String	*libNames	List of library cells.
unsigned int	numLibs	Number of allowed library cells.
String	*viewNames	List of view names.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

Type	Name	Description
unsigned int	numViews	Number of allowed view names.

Enumerated Types and Enumerated Constants used in Placement Infrastructure

This section covers the enumerated types and enumerated constants used in placement infrastructure.

dbAlignmentType

Alignment	Enumerated constants
No alignment	dbcUnknownAlignment
Align to bottom edge	dbcBottomAlignment
Align to top edge	dbcTopAlignment
Align to center point	dbcCenterAlignment
Align to left edge	dbcLeftAlignment
Align to right edge	dbcRightAlignment
Use any alignment	dbcAnyAlignmen

dbOrient

Allowed orientations	Enumerated constants
No rotation	dbcR0
Rotate 90 degrees	dbcR90
Rotate 180 degrees	dbcR180
Rotate 270 degrees	dbcR270
Mirror about the Y axis	dbcMY
Mirror about the Y axis, then rotate 90 degrees	dbcMYR90
Mirror about the X axis	dbcMX
Mirror about the X axis, then rotate 90 degrees	dbcMXR90

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbRailFigType

Allowed railFig Types	Enumerated constants
No figType	dbcUnknownRailFigType
Rectangle figType	dbcRectRailFigType
PathSeg figType	dbcPathSegRailFigType

dbPlaceRowType

Allowed placeRow Types	Enumerated constants
No rowType specified	dbcUnknownPlaceRowType
Place area type	dbcAreaPlaceRowType
Placement row type	dbcRowPlaceRowType

Background Definition Functions

- [dbAllocBackgroundDefAttrArray](#)
- [dbCreateBackgroundDef](#)
- [dbCreateBackgroundDefByAttr](#)
- [dbDeleteBackgroundDef](#)
- [dbCreateBackgroundDef](#)
- [dbGenBackgroundDef](#)
- [dbGetBackgroundDefAttr](#)
- [dbGetBackgroundDefEnclosure](#)
- [dbGetBackgroundDefLP](#)
- [dbGetBackgroundDefName](#)
- [dbGetBackgroundDefNetName](#)
- [dbSetBackgroundDefEnclosures](#)
- [dbSetBackgroundDefLP](#)
- [dbSetBackgroundDefNetName](#)
- [dbStartGenBackgroundDef](#)

dbAllocBackgroundDefAttrArray

```
dbBackgroundDefAttrData*  
dbAllocBackgroundDefAttrArray  
    unsigned int numBackgroundDefAttrDatas  
    );
```

Description

Allocates memory for the array of backgroundDef attribute data and initializes the memory with zeroes.

Arguments

numBackgroundDefAttrDatas

Number of backgroundDef attributes to fit in the allocated memory.

Return Value

*dbBackgroundDefAttrData**

Pointer to the allocated memory area.

For more information, see [dbBackgroundDefAttrData](#).

dbCreateBackgroundDef

dbBackgroundDefId

```
dbCreateBackgroundDef(  
    dbCellViewId  cellViewId,  
    constString   nameBackgroundDef,  
    dbLayer       layer,  
    dbPurpose     purpose,  
    constString   netName,  
    int           leftEnclosure,  
    int           rightEnclosure,  
    int           topEnclosure,  
    int           bottomEnclosure  
);
```

Description

Creates a backgroundDef based on the specified attributes. The layer-purpose pair must exist to create a backgroundDef.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>nameBackgroundDef</i>	Name of a backgroundDef object.
<i>layer</i>	Layer to be associated with the backgroundDef object.
<i>purpose</i>	Purpose to be associated with the backgroundDef object.
<i>netName</i>	Name of a net.
<i>leftEnc</i>	Left enclosure of the background shape.
<i>rightEnc</i>	Right enclosure of the background shape.
<i>topEnc</i>	Top enclosure of the background shape.
<i>bottomEnc</i>	Bottom enclosure of the background shape.

Return Value

<i>dbBackgroundDefId</i>	Database ID of the created backgroundDef object.

dbCreateBackgroundDefByAttr

```
dbBackgroundDefId  
dbCreateBackgroundDefByAttr(  
    dbCellViewId          cellViewId,  
    onstString            name,  
    dbBackgroundDefAttrData *data  
);
```

Description

Creates a backgroundDef based on specified attributes.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of a backgroundDef object.
<i>*data</i>	Struct with the backgroundDef attributes. For more information, see dbBackgroundDefAttrData .

Return Value

<i>dbBackgroundDefId</i>	Database ID of the backgroundDef object that is created.

dbDeleteBackgroundDef

Boolean

```
dbDeleteBackgroundDef(  
    dbBackgroundDefId bgDefId  
);
```

Description

Deletes a backgroundDef object with the specified database ID.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbFreeBackgroundDefAttrArray

```
void  
dbFreeBackgroundDefAttrArray(  
    unsigned int      numBackgroundDefAttrDatas,  
    dbBackgroundDefAttrData *backgroundDefAttrData  
);
```

Description

Deallocates the memory allocated for an array of backGroundDef objects.

Arguments

<i>numBackgroundDefAttrDatas</i>	Size of the deallocated memory area.
<i>backgroundDefAttrData</i>	Pointer to the memory area to deallocate. For more information, see dbBackgroundDefAttrData .

Return Value

No Return Value.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGenBackgroundDef

Boolean

```
dbGenBackgroundDef (
    dbGenStateId      pState,
    dbBackgroundDefId *bgDefId
);
```

Description

Retrieves the ID of the next backgroundDef object from the generator with the given *pState*. ID of the backgroundDef object is returned in *bgDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>bgDefId</i>	ID of the next backgroundDef object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if the next backgroundDef ID is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGetBackgroundDefAttr

Boolean

```
dbGetBackgroundDefAttr(  
    dbBackgroundDefId bgDefId,  
    dbBackgroundDefAttrData *data  
);
```

Description

Returns the attributes of the specified backgroundDef object.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.
<i>data</i>	Pointer to struct in which the background def attributes will be returned. For more information, see dbBackgroundDefAttrData .

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetBackgroundDefEnclosure

Boolean

```
dbGetBackgroundDefEnclosures (  
    dbBackgroundDefId  bgDefId,  
    int                 *pLeft,  
    int                 *pRight,  
    int                 *pTop,  
    int                 *pBottom  
);
```

Description

Returns all enclosures of the specified backgroundDef object.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.
<i>pLeft</i>	Pointer to left enclosure of the background shape.
<i>pRight</i>	Pointer to right enclosure of the background shape.
<i>pTop</i>	Pointer to top enclosure of the background shape.
<i>pBottom</i>	Pointer to bottom enclosure of the background shape.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetBackgroundDefLP

Boolean

```
dbGetBackgroundDefLP(  
    dbBackgroundDefId  bgDefId,  
    dbLayer            *pLayer,  
    dbPurpose          *pPurpose  
);
```

Description

Returns the layer and purpose of the specified backgroundDef object.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.
<i>pLayer</i>	Pointer to the layer number to be associated with the backgroundDef.
<i>pPurpose</i>	Pointer to the purpose number to be associated with the backgroundDef.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetBackgroundDefName

String

```
dbGetBackgroundDefName (  
    dbBackgroundDefId    bgDefId  
);
```

Description

Returns the name of the specified backgroundDef object.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.

Return Value

<i>String</i>	Name of the specified backgroundDef object.

dbGetBackgroundDefNetName

String

```
dbGetBackgroundDefNetName (  
    dbBackgroundDefId bgDefId  
);
```

Description

Returns the net name of a backgroundDef with the specified database ID.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.

Return Value

<i>String</i>	Name of the net of the specified backgroundDef object.

dbSetBackgroundDefEnclosures

Boolean

```
dbSetBackgroundDefEnclosures (
    dbBackgroundDefId  bgDefId,
    int                left,
    int                right,
    int                top,
    int                bottom
);
```

Description

Sets specified enclosures on a backgroundDef with the specified database ID.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef object.
<i>pLeft</i>	Left enclosure of the background shape.
<i>pRight</i>	Right enclosure of the background shape.
<i>pTop</i>	Top enclosure of the background shape.
<i>pBottom</i>	Bottom enclosure of the background shape.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetBackgroundDefLP

Boolean

```
dbSetBackgroundDefLP(  
    dbBackgroundDefId  bgDefId,  
    dbLayer             layer,  
    dbPurpose           purpose  
);
```

Description

Sets specified layer and purpose for a backgroundDef with the specified database ID.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef.
<i>pLayer</i>	Layer number to be associated with the backgroundDef.
<i>pPurpose</i>	Purpose number to be associated with the backgroundDef.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetBackgroundDefNetName

Boolean

```
dbSetBackgroundDefNetName (
    dbBackgroundDefId bgDefId,
    constString        netName
);
```

Description

Sets the given net name for a backgroundDef with the specified database ID.

Arguments

<i>bgDefId</i>	Database ID of a backgroundDef.
<i>netName</i>	Name of the net to be set for backgroundDef.

Return Value

<i>Boolean</i>	TRUE if the net name is set. FALSE if the net name is not set.

dbStartGenBackgroundDef

```
dbGenStateId  
dbStartGenBackgroundDef(  
    dbCellViewId    cellViewId  
);
```

Description

Initializes a generator for all backgroundDefs in the specified cellview. A generator state ID is returned, which is used with the [dbGenBackgroundDef](#) function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Component Type Set Definition Functions

- [dbAllocCompTypeSetDefAttrArray](#)
- [dbCreateCompTypeSetDef](#)
- [dbCreateCompTypeSetDefByAttr](#)
- [dbFreeCompTypeSetDefAttrArray](#)
- [dbGenCompTypeSetDef](#)
- [dbGetCompTypeSetDefAttr](#)
- [dbGetCompTypeSetDefCompFilters](#)
- [dbSetCompTypeSetDefCompFilters](#)
- [dbStartGenCompTypeSetDef](#)
- [dbUnsetCompTypeSetDefCompFilters](#)

dbAllocCompTypeSetDefAttrArray

```
dbCompTypeSetDefAttrData*  
dbAllocCompTypeSetDefAttrArray(  
    unsigned int numCompTypeSetDefAttrData  
);
```

Description

Allocates memory for the array of compTypeSetDef attribute data and initializes the memory with zeroes.

Arguments

numCompTypeSetDefAttrData

Number of compTypeSetDef attributes to fit in the allocated memory.

Return Value

*dbCompTypeSetDefAttrData**

Pointer to the allocated memory area.

dbCreateCompTypeSetDef

dbId

```
dbCreateCompTypeSetDef(  
    dbCellViewId cellViewId,  
    constString allowedCompName,  
    dbAlignmentType areaRef,  
    dbAlignmentType compRef,  
    dbOrient *orients,  
    int numOrients,  
    constString *compTypes,  
    int numCompTypes,  
    dbCoord offset,  
    dbCoord grid  
);
```

Description

Creates a compTypeSetDef object in the specified cellView.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>allowedCompName</i>	Name of the allowed component.
<i>areaRef</i>	Specifies area reference. For more information, see dbAlignmentType .
<i>compRef</i>	Specifies component reference. For more information, see dbAlignmentType .
<i>orients</i>	Array of allowed orientations that placer can use for cells in the component types. For more information, see dbOrient .
<i>numOrients</i>	Number of entries in the array of allowed orientations.
<i>compTypes</i>	Array of component types names.
<i>numCompTypes</i>	Number of entries in the array of component type names.
<i>offset</i>	The offset from the reference points computed from row reference can be specified in user units.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

grid

The pitch or grid to use between placement locations can be specified.

Return Value

dbCreateCompTypeSetDef

Database ID of the created compTypeSetDef object.

dbCreateCompTypeSetDefByAttr

```
dbCompTypeSetDefId  
dbCreateCompTypeSetDefByAttr(  
    dbCellViewId          cellViewId,  
    constString            name,  
    dbCompTypeSetDefAttrData *data  
);
```

Description

Creates a compTypeSetDef based on the specified attributes.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of a compTypeSetDef object.
<i>*data</i>	Pointer to the struct of compTypeSetDef attributes. For more information, see <u>dbCompTypeSetDefAttrData</u> .

Return Value

<i>dbCompTypeSetDefId</i>	Database ID of the created compTypeSetDef object.

dbFreeCompTypeSetDefAttrArray

void

```
dbFreeCompTypeSetDefAttrArray(  
    unsigned int          numCompTypeSetDefAttrData,  
    dbCompTypeSetDefAttrData *compTypeSetDefAttrDatas  
);
```

Description

Deallocates the memory allocated for an array of compTypeSetDef objects.

Arguments

numCompTypeSetDefAttrData

Size of the deallocated memory area.

compTypeSetDefAttrDatas Pointer to the memory area to deallocate.

Return Value

No Return Value.

dbGenCompTypeSetDef

Boolean

```
dbGenCompTypeSetDef (  
    dbGenStateId    pState,  
    dbId            *pCompTypeSetDefId  
);
```

Description

Retrieves the ID of the next compTypeSetDef object from the generator with the given *pState*. ID of the backgroundDef is returned in *pCompTypeSetDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>pCompTypeSetDefId</i>	ID of the next compTypeSetDef object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next compTypeSetDef object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGetCompTypeSetDefAttr

Boolean

```
dbGetCompTypeSetDefAttr(  
    dbCompTypeSetDefId compTypeSetDefId,  
    dbCompTypeSetDefAttrData *data  
);
```

Description

Returns the attributes of the specified compTypeSetDef object.

Arguments

<i>compTypeSetDefId</i>	Database ID of a compTypeSetDef object.
<i>data</i>	Pointer to the struct of compTypeSetDef attributes.
	For more information, see <u>dbCompTypeSetDefAttrData</u> .

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetCompTypeSetDefCompFilters

Boolean

```
dbGetCompTypeSetDefCompFilters (
    dbId compTypeSetDefId,
    **compFilters,
    int      *numCompFilters
);
```

Description

Returns the compFilters in the specified compTypeSetDef object.

Arguments

<i>compTypeSetDefId</i>	Database ID of a compTypeSetDef object.
<i>compFilters</i>	Pointer to the memory area in which the array of compFilters is placed. For more information, see dbCompFilterSpec .
<i>numCompFilters</i>	Pointer to the memory area in which the number of compFilters is placed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetCompTypeSetDefCompFilters

Boolean

```
dbSetCompTypeSetDefCompFilters (
    dbId compTypeSetDefId,
    dbCompFilterSpec *compFilters,
    int numCompFilters
);
```

Description

Sets the compFilters in the specified compTypeSetDef object.

Arguments

<i>compTypeSetDefId</i>	Database ID of a compTypeSetDef object.
<i>compFilters</i>	Array of compFilters to be set. For more information, see dbCompFilterSpec .
<i>numCompFilters</i>	Number of elements in the array of compFilters.

Return Value

<i>Boolean</i>	TRUE if the compFilters are set. FALSE if the compFilters are not set.

dbStartGenCompTypeSetDef

```
dbGenStateId  
    dbStartGenCompTypeSetDef(  
        dbCellViewId cellViewId  
    );
```

Description

Initializes a generator for all compTypeSetDef objects in the specified cellview. A generator state ID is returned, which is used with the dbGenCompTypeSetDef function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbUnsetCompTypeSetDefCompFilters

Boolean

```
dbUnsetCompTypeSetDefCompFilters (  
    dbId compTypeSetDefId  
);
```

Description

Removes the compFilters from the specified compTypeSetDef object.

Arguments

<i>compTypeSetDefId</i>	Database ID of a compTypeSetDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Rail Definition Functions

- [dbAllocRailDefAttrArray](#)
- [dbCreateRailDef](#)
- [dbCreateRailDefByAttr](#)
- [dbFreeRailDefAttrArray](#)
- [dbGenRailDef](#)
- [dbGetRailDefAttr](#)
- [dbGetRailDefFigType](#)
- [dbSetRailDefFigType](#)
- [dbStartGenRailDef](#)

dbAllocRailDefAttrArray

```
dbRailDefAttrData*  
dbAllocRailDefAttrArray(  
    unsigned int numRailDefAttrDatas  
);
```

Description

Allocates memory for the array of railDef attribute data and initializes the memory with zeroes.

Arguments

<i>numRailDefAttrDatas</i>	Number of railDef attributes to fit in the allocated memory.

Return Value

<i>dbRailDefAttrData</i>	Pointer to the allocated memory area.

dbCreateRailDef

dbId

```
dbCreateRailDef(  
    dbCellViewId cellViewId,  
    String railName,  
    String netName,  
    dbAlignmentType areaRef,  
    dbAlignmentType railRef,  
    dbLayer layer,  
    dbPurpose purpose,  
    dbDistance width,  
    dbCoord offset,  
    dbCoord beginOffset,  
    dbCoord endOffset  
);
```

Description

Creates a railDef in the specified cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>railName</i>	Name of a railDef object.
<i>netName</i>	Name of a net.
<i>areaRef</i>	Alignment type of area of the rail geometry. For more information, see dbAlignmentType .
<i>railRef</i>	Alignment type of reference of the rail geometry. For more information, see dbAlignmentType .
<i>layer</i>	Layer to be associated with the railDef object.
<i>purpose</i>	Purpose to be associated with the railDef object.
<i>width</i>	Width of rail geometry.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

<i>offset</i>	Places the rail 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>beginOffset</i>	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>endOffset</i>	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.

Return Value

<i>dbId</i>	Database ID of the created railDef object.

dbCreateRailDefByAttr

```
dbRailDefId  
dbCreateRailDefByAttr(  
    dbCellViewId      cellViewId,  
    constString        name,  
    dbRailDefAttrData *data  
);
```

Description

Creates a railDef object based on the specified attributes.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name for the railDef object to be created.
<i>*data</i>	Pointer to struct with railDef attributes.

Return Value

<i>dbRailDefId</i>	Database ID of the created railDef object.

dbFreeRailDefAttrArray

```
void  
dbFreeRailDefAttrArray(  
    unsigned int      numRailDefAttrDatas,  
    dbRailDefAttrData *railDefAttrDatas  
);
```

Description

Deallocates the memory allocated for an array of railDef objects.

Arguments

<i>numRailDefAttrDatas</i>	Size of the deallocated memory area.
<i>railDefAttrDatas</i>	Pointer to the memory area to deallocate.

Return Value

No Return Value.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGenRailDef

Boolean

```
dbGenRailDef(  
    dbGenStateId  pState,  
    dbId          *pRailDefId  
);
```

Description

Retrieves the ID of the next railDef object from the generator with the given *pState*. ID of the railDef is returned in *pRailDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>pRailDefId</i>	ID of the next railDef object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next railDef object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGetRailDefAttr

Boolean

```
dbGetRailDefAttr(  
    dbRailDefId      railDefId,  
    dbRailDefAttrData *data  
);
```

Description

Returns the attributes of the specified railDef object.

Arguments

<i>RailDefId</i>	Database ID of a railDef object.
<i>*data</i>	Pointer to the railDef attributes.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetRailDefFigType

```
dbRailFigType  
dbGetRailDefFigType(  
    dbRailDefId railDefId  
);
```

Description

Returns the rail figure type for the specified railDef object.

Arguments

<i>railDefId</i>	Database ID of a railDef object.

Return Value

<i>dbRailFigType</i>	Rail figure type for the specified railDefObject. For more information, see dbRailFigType .

dbSetRailDefFigType

Boolean

```
dbSetRailDefFigType(  
    dbRailDefId      railDefId,  
    dbRailFigType value  
);
```

Description

Sets the rail figure type for the specified railDef object.

Arguments

<i>railDefId</i>	Database ID of a railDef object.
<i>value</i>	Type of railFigType. For more information, see dbRailFigType .

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbStartGenRailDef

```
dbGenStateId  
dbStartGenRailDef(  
    dbCellViewId cellViewId  
);
```

Description

Initializes a generator for all railDef objects in the specified cellview. A generator state ID is returned, which is used with the [dbGenRailDef](#) function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Placement Area Definition Functions

- [dbAddPlaceAreaBackgroundDef](#)
- [dbAddPlaceAreaCompTypeSetDef](#)
- [dbAddPlaceAreaRailDef](#)
- [dbAllocPlaceRowAttrArray](#)
- [dbCreatePlaceArea](#)
- [dbFreePlaceRowAttrArray](#)
- [dbGenPlaceArea](#)
- [dbGenPlaceAreaToBackgroundDef](#)
- [dbGenPlaceAreaToCompTypeSetDef](#)
- [dbGenPlaceAreaToRailDef](#)
- [dbGetPlaceAreaOrient](#)
- [dbGetPlaceAreaUtilization](#)
- [dbRemovePlaceAreaBackgroundDef](#)
- [dbRemovePlaceAreaCompTypeSetDef](#)
- [dbRemovePlaceAreaRailDef](#)
- [dbSetPlaceAreaOrient](#)
- [dbSetPlaceAreaUtilization](#)
- [dbStartGenPlaceArea](#)
- [dbStartGenPlaceAreaToBackgroundDef](#)
- [dbStartGenPlaceAreaToCompTypeSetDef](#)
- [dbStartGenPlaceAreaToRailDef](#)

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbAddPlaceAreaBackgroundDef

Boolean

```
dbAddPlaceAreaBackgroundDef (  
    dbId  placeAreaId,  
    dbId  backgroundDef  
);
```

Description

Adds a reference from the placeArea object to the backgroundDef object. A warning message is displayed if the objects are not of the correct type.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>backgroundDef</i>	Database ID of a backgroundDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbAddPlaceAreaCompTypeSetDef

Boolean

```
dbAddPlaceAreaCompTypeSetDef(  
    dbId placeAreaId,  
    dbId compTypeSetDefId  
);
```

Description

Adds a reference from the placeArea object to the compTypeSetDef object. A warning message is displayed if the objects are not of the correct type.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>compTypeSetDefId</i>	Database ID of a compTypeSetDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbAddPlaceAreaRailDef

Boolean

```
dbAddPlaceAreaRailDef(  
    dbId placeAreaId,  
    dbId railDef  
);
```

Description

Adds a reference from the placeArea to the railDef. A warning message is displayed if the objects are not of the correct type.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>railDef</i>	Database ID of a railDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbAllocPlaceRowAttrArray

```
dbPlaceRowAttrData*  
dbAllocPlaceRowAttrArray(  
    unsigned int      numPlaceRowAttrs  
);
```

Description

Allocates memory for the array of placeRow attribute data and initializes the memory with zeroes.

Arguments

<i>numPlaceRowAttrs</i>	Number of placeRow attributes to fit in the allocated memory.

Return Value

<i>dbPlaceRowAttrData*</i>	Pointer to the allocated memory area.

dbCreatePlaceArea

dbId

```
dbCreatePlaceArea(  
    dbCellViewId cellViewId,  
    constString  name,  
    dbDistance  widthIn,  
    dbDistance  heightIn,  
    dbPoint     originIn,  
    dbOrient    orientIn,  
    double      utilization  
);
```

Description

Creates a custom placeArea object in the specified cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of the placeArea object.
<i>widthIn</i>	Width of the placeArea in user units.
<i>heightIn</i>	Height of the placeArea in user units.
<i>originIn</i>	Origin of the placeArea object.
<i>orientIn</i>	Orientation at the placeArea object. For more information, see dbOrient .
<i>utilization</i>	Specifies the utilization of the placeArea object.



This attribute is not being used in Virtuoso.

Return Value

<i>dbId</i>	Database ID of the created placeArea object.

dbFreePlaceRowAttrArray

```
void  
dbFreePlaceRowAttrArray(  
    unsigned int      numAttrs,  
    dbPlaceRowAttrData *attrs  
);
```

Description

Deallocates the memory allocated for an array of placeRow objects.

Arguments

<i>numBackgroundDefAttrDatas</i>	Size of the deallocated memory area.
<i>backgroundDefAttrData</i>	Pointer to the memory area to deallocate. For more information, see dbBackgroundDefAttrData .

Return Value

No Return Value.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGenPlaceArea

Boolean

```
dbGenPlaceArea(  
    dbGenStateId pState,  
    dbId         *pPlcId  
);
```

Description

Retrieves the ID of the next custom placeArea object from the generator with the given *pState*. ID of the placeArea is returned in *pPlcId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>pPlcId</i>	ID of the next placeArea object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next placeArea object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenPlaceAreaToBackgroundDef

Boolean

```
dbGenPlaceAreaToBackgroundDef(  
    bGenStateId      pState,  
    dbBackgroundDefId *bgDefId  
);
```

Description

Retrieves the ID of the next backgroundDef object for the placeArea object from the generator with the given *pState*. ID of the backgroundDef is returned in *bgDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>bgDefId</i>	ID of the next backgroundDef object for the placeArea object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next backgroundDef object for the placeArea object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenPlaceAreaToCompTypeSetDef

Boolean

```
dbGenPlaceAreaToCompTypeSetDef(  
    dbGenStateId pState,  
    dbId          *pCompTypeSetDefId  
);
```

Description

Retrieves the ID of the next allowed compTypeSetDef object for the placeArea object from the generator with the given *pState*. ID of the placeArea is returned in *pCompTypeSetDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>pCompTypeSetDefId</i>	ID of the next allowed compTypeSetDef object for the placeArea object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next compTypeSetDef object for the placeArea object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenPlaceAreaToRailDef

Boolean

```
dbGenPlaceAreaToRailDef(  
    dbGenStateId pState,  
    dbId *pRailDefId  
);
```

Description

Retrieves the ID of the next railDef object for the placeArea from the generator with the given *pState*. ID of the placeArea is returned in *pRailDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>pRailDefId</i>	ID of the next railDef object for the placeArea object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next railDef object for the placeArea object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGetPlaceAreaOrient

dbOrient

```
dbGetPlaceAreaOrient(  
    dbId placeAreaId  
);
```

Description

Returns the orient attribute for the specified custom placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.

Return Values

<i>dbOrient</i>	Orientation of the specified placeArea object. For more information, see dbOrient .

dbGetPlaceAreaUtilization

double

```
dbGetPlaceAreaUtilization(  
    dbId placeAreaId  
);
```

Description

Returns the the placement utilization attribute for the specified custom placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbRemovePlaceAreaBackgroundDef

Boolean

```
dbRemovePlaceAreaBackgroundDef(  
    dbId          placeAreaId,  
    dbBackgroundDefId bgDefId  
);
```

Description

Removes the specified backgroundDef from the specified placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>bgDefId</i>	Database ID of the backgroundDef object to be removed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbRemovePlaceAreaCompTypeSetDef

Boolean

```
dbRemovePlaceAreaCompTypeSetDef (  
    dbId placeAreaId,  
    dbId compTypeSetDef  
);
```

Description

Removes the specified compTypeSetDef object from the specified placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>compTypeSetDef</i>	Database ID of the compTypeSetDef object to be removed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbRemovePlaceAreaRailDef

Boolean

```
dbRemovePlaceAreaRailDef(  
    dbId placeAreaId,  
    dbId railDef  
);
```

Description

Removes the specified railDef from the specified placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>railDef</i>	Database ID of the railDef object to be removed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetPlaceAreaOrient

Boolean

```
dbSetPlaceAreaOrient(  
    dbId placeAreaId,  
    dbOrient orientIn  
);
```

Description

Sets the given orientation for the specified placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>orientIn</i>	Orientation to be associated with the placeArea object. For more information, see dbOrient .

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetPlaceAreaUtilization

Boolean

```
dbSetPlaceAreaUtilization(  
    dbId      placeAreaId,  
    double    utilization  
);
```

Description

Sets the specified placement utilization attribute for the specified custom placeArea object.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.
<i>utilization</i>	Placement utilization attribute to be associated with the placeArea object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbStartGenPlaceArea

```
dbGenStateId  
  
dbStartGenPlaceArea(  
    dbCellViewId  cellViewId  
);
```

Description

Initializes a generator for all placeArea objects in the specified cellview. A generator state ID is returned, which is used with the [dbGenPlaceArea](#) function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenPlaceAreaToBackgroundDef

```
dbGenStateId  
  
dbStartGenPlaceAreaToBackgroundDef(  
    dbId placeAreaId  
);
```

Description

Initializes a generator for all backgroundDef objects associated with the specified placeArea object. A generator state ID is returned, which is used with the [dbGenPlaceAreaToBackgroundDef](#) function.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenPlaceAreaToCompTypeSetDef

```
dbGenStateId  
dbStartGenPlaceAreaToCompTypeSetDef(  
    dbId placeAreaId  
);
```

Description

Initializes a generator for all compTypeSetDef objects associated with the specified placeArea object. A generator state ID is returned, which is used with the dbGenPlaceAreaToCompTypeSetDef function.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenPlaceAreaToRailDef

```
dbGenStateId  
dbStartGenPlaceAreaToRailDef(  
    dbId placeAreaId  
);
```

Description

Initializes a generator for all railDefs associated with the specified placeArea object. A generator state ID is returned, which is used with the [dbGenPlaceAreaToRailDef](#) function.

Arguments

<i>placeAreaId</i>	Database ID of a placeArea object.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Row-based Functions

A row object represents a row of sites where a cellview can be placed. A row represents a potential location for placement of standard cells, macros, or devices. A row is defined as an area associated with a siteDef. The siteDef has information allowing for the specification of the objects that can be placed within a row. A row is a repetition of a site element in a given direction (horizontal or vertical).

This section describes the following functions that affect row objects.

- [dbAddRowBackgroundDef](#)
- [dbAddRowCompTypeSetDef](#)
- [dbAddRowRailDef](#)
- [dbCreateRow](#)
- [dbDeleteRow](#)
- [dbFindRowByName](#)
- [dbGenRow](#)
- [dbGenRowHeaderToRow](#)
- [dbGenRowToBackgroundDef](#)
- [dbGenRowToCompTypeSetDef](#)
- [dbGenRowToRailDef](#)
- [dbGetRow](#)
- [dbStartGenRowToCompTypeSetDef](#)
- [dbGetRowName](#)
- [dbGetRowNumSites](#)
- [dbGetRowSiteOrient](#)
- [dbGetRowOrigin](#)
- [dbGetRowRowHeader](#)
- [dbGetRowSiteDef](#)
- [dbGetRowSiteDefName](#)
- [dbGetRowSiteOrient](#)

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

- [dbGetRowUtilization](#)
- [dbIsRowBound](#)
- [dbIsRowId](#)
- [dbRemoveRowBackgroundDef](#)
- [dbRemoveRowCompTypeSetDef](#)
- [dbRemoveRowCompTypeSetDef](#)
- [dbRemoveRowRailDef](#)
- [dbSetRowName](#)
- [dbSetRowNumSites](#)
- [dbSetRowOrient](#)
- [dbSetRowOrigin](#)
- [dbSetRowSiteDef](#)
- [dbSetRowSiteOrient](#)
- [dbSetRowUtilization](#)
- [dbStartGenRow](#)
- [dbStartGenRowHeaderToRow](#)
- [dbStartGenRowToBackgroundDef](#)
- [dbStartGenRowToRailDef](#)

dbAddRowBackgroundDef

Boolean

```
dbAddRowBackgroundDef (  
    dbRowId rowId  
    dbId     backgroundDef  
);
```

Description

Adds a reference from the row object to the backgroundDef object. A warning message is displayed if the objects are not of the correct type.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>backgroundDef</i>	Database ID of a backgroundDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbAddRowCompTypeSetDef

Boolean

```
dbAddRowCompTypeSetDef (
    dbRowId    rowId
    dbId       compTypeSetDefId
);
```

Description

Adds the specified compTypeSetDef object to the specified row object. A warning message is displayed if the objects are not of the correct type.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>compTypeSetDefId</i>	Database ID of a compTypeSetDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbAddRowRailDef

Boolean

```
dbAddRowRailDef(  
    dbRowId      rowId  
    dbId         railDefId  
);
```

Description

Adds the specified placement railDef to the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>railDefId</i>	Database ID of a railDef object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbCreateRow

dbRowId

```
dbCreateRow(  
    dbCellViewId      cellViewId,  
    techSiteDefId     siteDefId,  
    String             name,  
    dbPoint            origin,  
    dbOrient           orient,  
    dbOrient           siteOrient,  
    uint32_t           numSites  
);
```

Description

Creates a row object with the given attributes in the specified cellview.

Arguments

<i>cellViewId</i>	Database of a cellview.
<i>siteDefId</i>	ID of siteDef to be associated with this row object.
<i>name</i>	Name of the row.
<i>origin</i>	Origin field at the row object.
<i>orient</i>	Orientation at the row object.
<i>siteOrient</i>	Orientation of each site in the row object.
<i>numSites</i>	Number of site placements in the row object.

Return Values

<i>dbRowId</i>	Database ID of the created row object. Otherwise, returns NULL.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbDeleteRow

Boolean

```
dbDeleteRow(  
    dbRowId      rowId  
);
```

Description

Deletes specified the row object.

Arguments

<i>rowId</i>	Database ID of the row object.

Return Values

<i>Boolean</i>	TRUE if the row object is successfully deleted. FALSE, if operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbFindRowByName

```
dbRowId  
dbFindRowByName (  
    dbCellViewId      cellViewId,  
    String             rowName  
);
```

Description

Searches for the specified row object if a row with the given row name exists in the specified cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>rowName</i>	Name of the row object to be searched.

Return Values

<i>dbRowId</i>	If the object is found, it returns the ID of the row. Otherwise, returns dbcNullId.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGenRow

Boolean

```
dbGenRow(  
    dbGenStateId      pState,  
    dbRowId           *pRowId  
);
```

Description

Retrieves the ID of the next row object from the generator with the given *pState*. ID of the row is returned in *pRowId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>*pRowId</i>	ID of the next row object produced by the generator. This value is undefined if FALSE is returned.

Return Values

<i>Boolean</i>	TRUE if ID of the next row object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenRowHeaderToRow

Boolean

```
dbGenRowHeaderToRow(  
    dbGenStateId      pState,  
    dbRowId           *pRowId  
);
```

Description

Retrieves the ID of the next row object with the given rowHeader from the generator with the given *pState*. ID of the row is returned in *pRowId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>*pRowId</i>	ID of the next row with the given rowHeader produced by the generator. This value is undefined if FALSE is returned.

Return Values

<i>Boolean</i>	TRUE if ID of the next row object with the given rowHeader is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenRowToBackgroundDef

Boolean

```
dbGenRowToBackgroundDef (  
    dbGenStateId      pState,  
    dbBackgroundDefId *bgDefId  
);
```

Description

Retrieves the ID of the next backgroundDef object for the specified row object from the generator with the given *pState*. ID of the backgroundDef is returned in *bgDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>bgDefId</i>	ID of the next backgroundDef object for the specified row object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next backgroundDef for the specified row object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenRowToCompTypeSetDef

Boolean

```
dbGenRowToCompTypeSetDef(  
    dbGenStateId pState,  
    dbId *pCompTypeSetDefId  
);
```

Description

Retrieves the ID of the next the allowed compTypeSetDef object associated with the specified row object from the generator with the given *pState*. ID of the backgroundDef object is returned in *pCompTypeSetDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>bgDefId</i>	ID of the next allowed compTypeSetDef object associated with the specified row object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next allowed compTypeSetDef object associated with the specified row object is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenRowToRailDef

Boolean

```
dbGenRowToRailDef(  
    dbGenStateId    pState  
    dbId            *pRailDefId  
);
```

Description

Retrieves the ID of the next railDef object associated with the specified row object from the generator with the given *pState*. ID of the backgroundDef is returned in *pRailDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>pRailDefId</i>	ID of the next railDef object associated with the specified row object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if ID of the next railDef object associated with the specified row object is successfully generated. FALSE if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRow

Boolean

```
dbGetRow(  
    dbRowId          rowId,  
    String            *rowName,  
    dbPoint           *origin  
    dbOrient          *orient,  
    dbSiteDefId       *pSiteDefId,  
    dbOrient          *siteOrient,  
    uint32_t          *numSites  
);
```

Description

Returns the attributes for the specified row object in the output parameters *rowName*, *origin*, *orient*, *pSiteDefId*, *siteOrient* and *numSites*.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>*rowName</i>	Pointer to the name of the row object.
<i>*origin</i>	Pointer to the origin point for the row object.
<i>*orient</i>	Pointer to the orientation attribute for the row object.
<i>*pSiteDefId</i>	Pointer to the siteDef associated with the row object.
<i>*siteOrient</i>	Pointer to the orientation of the site which is associated with the row object.
<i>*numSites</i>	Pointer to the number of sites for the row object.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbStartGenRowToCompTypeSetDef

```
dbGenStateId  
dbStartGenRowToCompTypeSetDef (  
    dbRowId rowId  
);
```

Description

Initializes a generator for all compTypeSetDef objects associated with the specified row object. A generator state ID is returned, which is used with the dbGenRowToCompTypeSetDef function.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRowName

String

```
dbGetRowName (
    dbRowId      rowId
);
```

Description

Returns the `name` for the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Values

<i>String</i>	Name for the row object.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRowNumSites

Boolean

```
dbGetRowNumSites (
    dbRowId      rowId,
    uint32_t*    numSites
);
```

Description

Returns the number of sites for the row object specified by *rowId* via the output parameter *numSites*.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>numSites</i>	Pointer to the number of sites for the row object.

Return Values

<i>Boolean</i>	TRUE if the number of sites for the row object specified is successfully returned.
	FALSE if the operation fails.

dbGetRowSiteOrient

Boolean

```
dbGetRowSiteOrient(  
    dbRowId      rowId,  
    dbOrient*    rowSiteOrient  
);
```

Description

Returns the site orientation of the specified row.

Arguments

<i>rowId</i>	Database ID for the row object.
<i>rowSiteOrient</i>	Pointer to the site orientation for the row object.

Return Values

<i>Boolean</i>	TRUE if the site orientation for the specified row is successfully returned. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRowOrient

Boolean

```
dbGetRowOrient(  
    dbRowId      rowId,  
    dbOrient*    orient  
);
```

Description

Returns the orient attribute for the row specified in *rowId*.

Arguments

<i>rowId</i>	Database ID of a row object .
<i>orient</i>	Pointer to the orientation attribute for the row.

Return Values

<i>Boolean</i>	TRUE if the orient for the row specified in <i>rowId</i> was successfully returned. FALSE if the operation fails.

dbGetRowOrigin

Boolean

```
dbGetRowOrigin(  
    dbRowId      rowId,  
    dbPoint*     rowOrigin  
);
```

Description

Returns the origin for the specified row.

Arguments

<i>rowId</i>	Database of a row object.
<i>rowOrigin</i>	Pointer to the origin for the row object.

Return Values

<i>Boolean</i>	TRUE if the origin for the specified row is successfully returned. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRowRowHeader

```
dbRowHeaderId  
dbGetRowRowHeader (  
    dbRowId      rowId  
);
```

Description

Returns the rowHeader associated with the specified row.

Arguments

<i>rowId</i>	Database of a row object.

Return Values

<i>dbRowHeaderId</i>	Returns the rowHeader associated for the specified row.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRowSiteDef

```
techSiteDefId  
dbGetRowSiteDef(  
    dbRowId      rowId  
);
```

Description

Returns the siteDef associated with the specified row.

Arguments

<i>rowId</i>	Database ID for a row object.

Return Values

<i>techSiteDefId</i>	Returns the siteDef for the specified row.

dbGetRowSiteDefName

String

```
dbGetRowSiteDefName (  
    dbRowId      rowId  
);
```

Description

Returns the name of the siteDef associated with the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Values

<i>String</i>	Returns the name of the siteDef for the specified row object.

dbGetRowSiteOrient

Boolean

```
dbGetRowSiteOrient(  
    dbRowId          rowId,  
    dbOrient*        rowSiteOrient  
);
```

Description

Returns the site orientation of the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>rowSiteOrigin</i>	Pointer to the site orientation for the row object.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbGetRowUtilization

```
double  
dbGetRowUtilization(  
    dbRowId    rowId  
);
```

Description

Returns the placement utilization attribute for the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Values

<i>double</i>	Placement utilization attribute for the specified row.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbIsRowBound

Boolean

```
dbIsRowBound(  
    dbRowId      rowId  
);
```

Description

Checks if the siteDef associated with the row is currently bound to the master.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Values

<i>Boolean</i>	TRUE if the siteDef associated with the row is bound to the master. FALSE if the siteDef associated with the row is not bound to the master.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbIsRowId

Boolean

```
dbIsRowId(  
    dbRowId      rowId  
);
```

Description

Verifies that the specified rowId refers to a valid row object.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Values

<i>Boolean</i>	TRUE if the specified rowId refers to a valid row object. FALSE if the specified rowId does not refers to a valid row object.

dbRemoveRowBackgroundDef

Boolean

```
dbRemoveRowBackgroundDef (  
    dbRowId rowId,  
    dbId     backgroundDef  
);
```

Description

Removes the specified placement backgroundDef from the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>attribute</i>	Database ID of the backgroundDej to be removed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbRemoveRowCompTypeSetDef

Boolean

```
dbRemoveRowCompTypeSetDef (  
    dbRowId      rowId,  
    dbId         dbCompTypeSetDefId  
);
```

Description

Removes the specified placement compTypeSetDef from the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>dbCompTypeSetDefId</i>	Database ID of the compTypeSetDef to be removed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbRemoveRowRailDef

Boolean

```
dbRemoveRowRailDef(  
    dbRowId    rowId,  
    dbId       railDefId  
);
```

Description

Removes the specified placement railDef from the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>railDefId</i>	Database ID of the railDef to be removed.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetRowName

Boolean

```
dbSetRowName (
    dbRowId      rowId,
    String       rowName
);
```

Description

Sets given name for the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>rowName</i>	Name to be given to the specified row.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetRowNumSites

Boolean

```
dbSetRowNumSites (
    dbRowId      rowId,
    uint32_t      numSites
);
```

Description

Sets the number of sites for the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>numSites</i>	Number of sites associated with the row to be set for the row object.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetRowOrient

Boolean

```
dbSetRowOrient(  
    dbRowId      rowId,  
    dbOrient      orient  
);
```

Description

Sets the *orient* attribute for the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>orient</i>	Orientation of the row. For more information, see dbOrient .

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetRowOrigin

Boolean

```
dbSetRowOrigin(  
    dbRowId      rowId,  
    dbPoint      origin  
);
```

Description

Sets the origin attribute for the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>origin</i>	Origin point of the row object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbSetRowSiteDef

Boolean

```
dbSetRowSiteDef(  
    dbRowId          rowId,  
    techSiteDefId    siteDefId  
);
```

Description

Sets the siteDef for the specified row object.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>siteDefId</i>	Database ID for the siteDef to be set.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetRowSiteOrient

Boolean

```
dbSetRowOrient(  
    dbRowId      rowId,  
    dbOrient     siteOrient  
);
```

Description

Sets the orientation of the site associated with the specified row object.

Arguments

<i>rowId</i>	The database <code>dbRowId</code> for the row object.
<i>siteOrient</i>	The orientation of the site associated with the row.

Return Values

Boolean	TRUE if the site orientation for <code>rowId</code> is successfully set. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbSetRowUtilization

Boolean

```
dbSetRowUtilization(  
    dbRowId    rowId  
    double     utilization  
);
```

Description

Sets the placement utilization attribute for the specified row.

Arguments

<i>rowId</i>	Database ID of a row object.
<i>utilization</i>	Placement utilization attribute to be associated with the row object. Valid values: Number between 0 and 1. Not used by Virtuoso.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

dbStartGenRow

```
dbGenStateId  
dbStartGenRow(  
    dbCellViewId    cellViewId  
);
```

Description

Initializes a generator for all rows in the specified cellview. A generator state ID is returned, which is used with the [dbGenRow](#) function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenRowHeaderToRow

```
dbGenStateId  
dbStartGenRowHeaderToRow(  
    dbRowHeaderId    rowHeaderId  
);
```

Description

Initializes a generator for all rows associated with the specified rowHeader. A generator state ID is returned, which is used with the [dbGenRowToBackgroundDef](#) function.

Arguments

<i>rowHeaderId</i>	Database ID of a rowHeader.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenRowToBackgroundDef

```
dbGenStateId  
dbStartGenRowToBackgroundDef (  
    dbRowId    rowId  
);
```

Description

Initializes a generator for all backgroundDefs associated with the specified row. A generator state ID is returned, which is used with the [dbGenRowToBackgroundDef](#) function.

Arguments

<i>placeAreaId</i>	Database ID of a row.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenRowToRailDef

```
dbGenStateId  
dbStartGenRowToRailDef(  
    dbRowId    rowId  
);
```

Description

Initializes a generator for all railDefs associated with the specified row object. A generator state ID is returned, which is used with the [dbGenRowToRailDef](#) function.

Arguments

<i>rowId</i>	Database ID of a row object.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Row Region Specification Functions

- [dbCreateRowRegionSpec](#)
- [dbDeleteRowRegionSpec](#)
- [dbFindRowRegionSpec](#)
- [dbGenRowRegionSpecs](#)
- [dbGenRowRegionSpecToRowRegion](#)
- [dbGetRowRegionSpecGrids](#)
- [dbGetRowRegionSpecHeight](#)
- [dbGetRowRegionSpecName](#)
- [dbGetRowRegionSpecOffsets](#)
- [dbGetRowRegionSpecRelSnapPattern](#)
- [dbGetRowRegionSpecRowAttrs](#)
- [dbSetRowRegionSpecGrids](#)
- [dbSetRowRegionSpecHeight](#)
- [dbSetRowRegionSpecOffset](#)
- [dbSetRowRegionSpecRelSnapPattern](#)
- [dbSetRowRegionSpecRowAttrs](#)
- [dbStartGenRowRegionSpecs](#)
- [dbStartGenRowRegionSpecToRowRegion](#)

dbCreateRowRegionSpec

dbRowRegionSpecId

```
dbCreateRowRegionSpec (
    dbCellViewId          cellViewId,
    constString            name,
    unsigned int           height,
    dbPlaceRowAttrData     *placeRowAttrs,
    unsigned int           numPlaceRowAttrs,
    constString            xRefName,
    constString            yRefName,
    int                    xOffset,
    int                    yOffset,
    constString            relSPName
);
```

Description

Creates a rowRegionSpec in the specified cellview.

A rowRegionSpec describes a list of rows, where each row can be either a dbRow or a dbPlaceArea:

- dbRow - Needs siteDef from techDB
- dbPlaceArea - Does not need a siteDef from the techDB

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of the rowRegionSpec.
<i>height</i>	Height of one complete instance of the rowRegionSpec used to create one row.
<i>placeRowAttrs</i>	Array of structs with the placement row attributes. Each array entry describes one row. For more information, see dbPlaceRowAttrData .
<i>numPlaceRowAttrs</i>	PlaceRowAttrData array.
<i>xRefName</i>	Name of reference grid in the X direction
<i>yRefName</i>	Name of reference grid in the Y direction
<i>xOffset</i>	List of the offsets from the reference grids in X direction.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

yOffset List of the offsets from the reference grids in Y direction.

relSPName Name of relatedSnapPattern.

Return Value

dbRowRegionSpecId Database ID of the rowRegionSpec that is created.

dbDeleteRowRegionSpec

Boolean

```
dbDeleteRowRegionSpec (  
    dbRowRegionSpecId rowRegionSpec  
);
```

Description

Deletes a rowRegionSpec object with the specified database ID.

Arguments

<i>rowRegionSpec</i>	Database ID of a rowRegionSpec object.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbFindRowRegionSpec

```
dbRowRegionSpecId  
dbFindRowRegionSpec(  
    dbCellViewId cellViewId,  
    constString name  
);
```

Description

Searches for a rowRegionSpec with the specified name in the given cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of a rowRegionSpec.

Return Value

<i>dbRowRegionSpecId</i>	Database ID of the searched rowRegionSpec, if found. Otherwise, returns dbcNullId.

dbGenRowRegionSpecs

Boolean

```
dbGenRowRegionSpecs (  
    dbGenStateId stateId,  
    dbRowRegionSpecId *objId  
);
```

Description

Retrieves the ID of the next rowRegionSpec object from the generator with the given *stateId*. ID of the rowRegionSpec object is returned in *objId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>objId</i>	ID of the next rowRegionSpec object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if the next rowRegionSpec ID is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenRowRegionSpecToRowRegion

Boolean

```
dbGenRowRegionSpecToRowRegion(  
    dbGenStateId  stateId,  
    dbRowRegionId *objId  
);
```

Description

Retrieves the ID of the next rowRegion object associated with a given rowRegionSpec from the generator with the given *stateId*. ID of the rowRegion object is returned in *objId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>objId</i>	ID of the next rowRegion object associated with a given rowRegionSpec produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if the next rowRegion ID is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGetRowRegionSpecGrids

Boolean

```
dbGetRowRegionSpecGrids(  
    dbRowRegionSpecId objId,  
    String              *refGridX,  
    String              *refGridY  
);
```

Description

Returns the reference grid of the specified rowRegionSpec in the X and Y direction.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>*refGridX</i>	Pointer to the reference grid of the rowRegionSpec in X direction.
<i>*refGridY</i>	Pointer to the reference grid of the rowRegionSpec in Y direction

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetRowRegionSpecHeight

`unsigned int`

```
dbGetRowRegionSpecHeight(  
    dbRowRegionSpecId objId  
);
```

Description

Returns the height of the specified rowRegionSpec.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.

Return Values

<i>unsigned int</i>	Height of the rowRegionSpec object.

dbGetRowRegionSpecName

String

```
dbGetRowRegionSpecName (  
    dbRowRegionSpecId objId  
);
```

Description

Returns the name of the specified rowRegionSpec.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.

Return Values

<i>String</i>	Name of the rowRegionSpec object.

dbGetRowRegionSpecOffsets

Boolean

```
dbGetRowRegionSpecOffsets (
    dbRowRegionSpecId objId,
    int                *offsetX,
    int                *offsetY
);
```

Description

Returns the reference grid offset of the specified rowRegionSpec in the X and Y direction.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>*offsetX</i>	Pointer to the reference grid offset of the rowRegionSpec in X direction.
<i>*offsetY</i>	Pointer to the reference grid offset of the rowRegionSpec in Y direction

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetRowRegionSpecRelSnapPattern

Boolean

```
dbGetRowRegionSpecRelSnapPattern(  
    dbRowRegionSpecId objId,  
    String              *relSnapPattern  
);
```

Description

Returns the relatedSnapPattern name of the specified rowRegionSpec.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>*relSnapPattern</i>	Pointer to the relatedSnapPattern name of the rowRegionSpec.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetRowRegionSpecRowAttrs

Boolean

```
dbGetRowRegionSpecRowAttrs(  
    dbRowRegionSpecId  objId,  
    dbPlaceRowAttrData **attrArray,  
    unsigned int       *numAttrs  
);
```

Description

Returns the placement row attributes of the specified rowRegionSpec.

Note: Use the [dbFreePlaceRowAttrArray](#) function to free the memory the CDB allocates for the attrArray.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>attrArray</i>	Pointer to the memory area where the array of placement row attributes is placed.
<i>*numAttrs</i>	Number of entries in the array of placement row attributes.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbSetRowRegionSpecGrids

Boolean

```
dbSetRowRegionSpecGrids(  
    dbRowRegionSpecId objId,  
    constString       refGridX,  
    constString       refGridY  
);
```

Description

Sets the reference grid for the specified rowRegionSpec in the X and Y direction.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>refGridX</i>	Pointer to the reference grid of the rowRegionSpec in X direction.
<i>refGridY</i>	Pointer to the reference grid of the rowRegionSpec in Y direction

Return Values

<i>Boolean</i>	TRUE if the reference grid is set for the specified rowRegionSpec in the X and Y direction. FALSE if the operation fails.

dbSetRowRegionSpecHeight

Boolean

```
dbSetRowRegionSpecHeight(  
    dbRowRegionSpecId objId,  
    unsigned int       height  
);
```

Description

Sets the height for the specified rowRegionSpec.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>unsigned int</i>	Height that needs to be set for the rowRegionSpec object.

Return Values

<i>Boolean</i>	TRUE if the height is set for the specified rowRegionSpec in the X and Y direction. FALSE if the operation fails.

dbSetRowRegionSpecOffset

Boolean

```
dbSetRowRegionSpecOffset(  
    dbRowRegionSpecId objId,  
    int                offsetX,  
    int                offsetY  
);
```

Description

Sets the reference grid offset for the specified rowRegionSpec in the X and Y direction.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>offsetX</i>	Reference grid offset of the rowRegionSpec in X direction.
<i>offsetY</i>	Reference grid offset of the rowRegionSpec in Y direction

Return Values

<i>Boolean</i>	TRUE if the reference grid offset is set for the specified rowRegionSpec in the X and Y direction. FALSE if the operation fails.

dbSetRowRegionSpecRelSnapPattern

Boolean

```
dbSetRowRegionSpecRelSnapPattern(  
    dbRowRegionSpecId objId,  
    constString       relSnapPattern  
);
```

Description

Sets the relatedSnapPattern name for the specified rowRegionSpec.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>relSnapPattern</i>	Name of relatedSnapPattern to be set for the rowRegionSpec.

Return Values

<i>Boolean</i>	TRUE if relatedSnapPattern is set for the specified rowRegionSpec. FALSE if the operation fails.

dbSetRowRegionSpecRowAttrs

Boolean

```
dbSetRowRegionSpecRowAttrs(  
    dbRowRegionSpecId  objId,  
    dbPlaceRowAttrData *attrArray,  
    unsigned int        numAttrs  
);
```

Description

Sets the placement row attributes for the specified rowRegionSpec.

Note: Use the dbAllocPlaceRowAttrDataArray function to allocate the attribute array to fill it before calling this function. You will also need to use [dbAllocRailDefAttrArray](#), [dbAllocBackgroundDefAttrArray](#), and [dbAllocCompTypeSetDefAttrArray](#) to fill and allocate the respective arrays with information. The [dbFreePlaceRowAttrArray](#) function should be used to free memory allocated to the array.

Arguments

<i>objId</i>	Database ID of a rowRegionSpec object.
<i>attrArray</i>	Pointer to the memory area where the array of placement row attributes is placed.
<i>*numAttrs</i>	Pointer to the memory area where the number of placement row attributes is placed.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbStartGenRowRegionSpecs

```
dbGenStateId  
dbStartGenRowRegionSpecs (  
    dbCellViewId cellViewId  
);
```

Description

Initializes a generator for all rowRegionSpec objects in the specified cellview. A generator state ID is returned, which is used with the [dbGenRowRegionSpecs](#) function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenRowRegionSpecToRowRegion

```
dbGenStateId  
dbStartGenRowRegionSpecToRowRegion(  
    dbRowRegionSpecId specId  
);
```

Description

Initializes a generator for all rowRegion objects associated with the specified rowRegionSpec. A generator state ID is returned, which is used with the [dbGenRowRegionSpecToRowRegion](#) function.

Arguments

<i>rowId</i>	Database ID of a rowRegionSpec object.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbGetRowRegionSpecPlaceCells

Boolean

```
dbGetRowRegionSpecPlaceCells(  
    dbRowRegionSpecId specId,  
    dbPlaceCellsAttrData *attrs  
);
```

Description

Returns the placeCells attributes of the specified rowRegionSpec.

Arguments

<i>specId</i>	Database ID of a rowRegionSpec object.
<i>*attrs</i>	Pointer to the placeCells attribute name of the rowRegionSpec. For more information, see dbPlaceCellsAttrData .

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the placeCells attribute is not set or if the operation fails.

dbSetRowRegionSpecPlaceCells

Boolean

```
dbSetRowRegionSpecPlaceCells(  
    dbRowRegionSpecId specId,  
    dbPlaceCellsAttrData *attrs  
);
```

Description

Sets the placeCells attributes for the specified rowRegionSpec.

Arguments

<i>specId</i>	Database ID of a rowRegionSpec object.
<i>*attrs</i>	Pointer to the placeCells attribute name of the rowRegionSpec.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbFreePlaceCellsListAttr

```
void  
dbFreePlaceCellsListAttr(  
    dbPlaceCellsAttrData *attrs  
);
```

Description

Deallocates the memory allocated for an array of placeCells objects.

Arguments

<i>*attrs</i>	Pointer to the memory area to deallocate.

Return Value

None

Row Region Functions

A rowRegion is a figGroup that is created and edited on the canvas. It references a rowRegionSpec.

A rowRegion should match its corresponding rowRegionSpec. However, if the rowRegionSpec is modified, all the rowRegions that reference it will no longer match their specification. In this case, you need to use the [dbRebuildRowRegion](#) function to rebuild the rowRegion to match its specification.

- [dbAttachRowRegionToPRBoundary](#)
- [dbCreateRowRegion](#)
- [dbDetachPRBoundaryFromRowRegion](#)
- [dbFindRowRegion](#)
- [dbFlattenRowRegion](#)
- [dbGenRowRegionPlaceRows](#)
- [dbGenRowRegions](#)
- [dbGetRowRegionAttachedToPRBoundary](#)
- [dbGetRowRegionPoints](#)
- [dbGetRowRegionRowRegionSpec](#)
- [dbGetRowRegionUsesPartialRowSpec](#)
- [dbIsRowRegion](#)
- [dbIsRowRegionAttachedToPRBoundary](#)
- [dbRebuildRowRegion](#)
- [dbRowRegionMatchesSpec](#)
- [dbSetRowRegionPoints](#)
- [dbSetRowRegionRowRegionSpec](#)
- [dbSetRowRegionUsesPartialRowSpec](#)
- [dbStartGenRowRegionPlaceRows](#)
- [dbStartGenRowRegions](#)

dbAttachRowRegionToPRBoundary

Boolean

```
dbAttachRowRegionToPRBoundary(  
    dbRowRegionId    rowRegionId  
);
```

Description

Attaches the PRBoundary to the specified rowRegion. Error occurs if the containing cellview does not have a PRBoundary.

Note: The PRBoundary is first detached from any rowRegion it might be attached to before being attached to the specified rowRegion.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.

Return Values

<i>Boolean</i>	TRUE if the PRBoundary is attached to the specified rowRegion. FALSE if the operation fails.

dbCreateRowRegion

```
dbRowRegionId
dbCreateRowRegion(
    dbCellViewId      cellViewId,
    constString       name,
    dbRowRegionSpecId rowRegionSpecId,
    unsigned int      numPoints,
    dbPoint           *pointArray,
    Boolean           isPartialRowSpec,
    dbOrient          orientation
);
```

Description

Creates a rowRegion object in the specified cellview.

Arguments.

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of the rowRegion object.
<i>rowRegionSpecId</i>	Database ID of the rowRegionSpec object to be used to create the rowRegion object.
<i>numPoints</i>	Number of points in pointArray that define the areaBoundary enclosing the rowRegion.
<i>pointArray</i>	Pointer to the point array.
<i>isPartialRowSpec</i>	Boolean specifying whether the fill space of rowRegion should be populated with partial specialization of rowRegionSpec.
<i>orientation</i>	Orientation of the rowRegion object. For more information, see dbOrient .

Return Value

<i>dbRowRegionId</i>	Database ID of the rowRegion that is created.

dbRowRegionPointsCutOut

Boolean

```
dbRowRegionPointsCutOut(  
    dbRowRegionId    rowRegionId,  
    dbBBox            *box  
);
```

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>rowRegionId</i>	Database ID of a rowRegion.
<i>box</i>	The box to cut out from the current rowRegion.

Value Returned

Boolean	True if the rowRegion points were modified successfully. False if the operation fails.

dbDetachPRBoundaryFromRowRegion

Boolean

```
dbDetachPRBoundaryFromRowRegion(  
    dbPRBoundaryId  prBoundaryId  
);
```

Description

Removes the rowRegion attached to the specified PRBoundary. Error occurs if the no rowRegion is attached to the PRBoundary.

Arguments

<i>prBoundaryId</i>	Database ID of a PRBoundary.

Return Values

<i>Boolean</i>	TRUE if the rowRegion is removed from the specified PRBoundary. FALSE if the operation fails.

dbFindRowRegion

```
dbRowRegionId  
dbFindRowRegion(  
    dbCellViewId  cellViewId,  
    constString   name  
);
```

Description

Searches for a rowRegion with the specified name in the given cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.
<i>name</i>	Name of a rowRegionSpec.

Return Value

<i>dbRowRegionId</i>	Database ID of the searched rowRegion, if found. Otherwise, returns dbcNullId.

dbFlattenRowRegion

Boolean

```
dbFlattenRowRegion(  
    dbId rowRegionId  
);
```

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>rowRegionId</i>	Database ID of the rowRegion to be flattened.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGenRowRegionPlaceRows

Boolean

```
dbGenRowRegionPlaceRows (  
    dbGenStateId    stateId,  
    dbId            *placeRowId  
);
```

Description

Retrieves ID of the next placement row associated with the rowRegion from the generator with the given *stateId*. ID of the rowRegion object is returned in *placeRowId*.

Arguments

<i>stateId</i>	State ID for the generator.
<i>placeRowId</i>	ID of the next placeRow object associated with a given rowRegion produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if the next rowRegion ID is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGenRowRegions

Boolean

```
dbGenRowRegions (
    dbGenStateId stateId,
    dbId          *rowRegionId
);
```

Description

Retrieves the ID of the next rowRegion object from the generator with the given *stateId*. ID of the rowRegion object is returned in *rowRegionId*.

Arguments

<i>stateId</i>	State ID for the generator.
<i>rowRegionId</i>	ID of the next rowRegion object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE if the next rowRegion ID is successfully generated. FALSE if the list is exhausted or an error occurs.

dbGetRowRegionAttachedToPRBoundary

```
dbRowRegionId  
dbGetRowRegionAttachedToPRBoundary(  
    dbPRBoundaryId prBoundaryId  
);
```

Description

Returns the rowRegion object that is attached to the specified PRBoundary.

Arguments

<i>prBoundaryId</i>	Database ID of a PRBoundary.
<i>*refGridX</i>	Pointer to the reference grid of the rowRegionSpec in X direction.
<i>*refGridY</i>	Pointer to the reference grid of the rowRegionSpec in Y direction

Return Values

<i>dbRowRegionId</i>	Database ID of the rowRegion object that is attached to the specified PRBoundary. Otherwise, returns NULL.

dbGetRowRegionPoints

Boolean

```
dbGetRowRegionPoints(  
    dbId          rowRegionId,  
    unsigned int  *numPoints,  
    dbPoint       **pointArray  
);
```

Description

Returns the points that define the areaBoundary enclosing the rowRegion

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.
<i>*numPoints</i>	Number of points in pointArray.
<i>**pointArray</i>	Pointer to the point array.

Return Value

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbGetRowRegionRowRegionSpec

```
dbRowRegionSpecId  
dbGetRowRegionRowRegionSpec (  
    dbId rowRegionId  
);
```

Description

Returns the rowRegionSpec from which the specified rowRegion was generated.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.

Return Value

<i>dbRowRegionSpecId</i>	Database ID of the rowRegionSpec from which the specified rowRegion was generated.

dbGetRowRegionUsesPartialRowSpec

Boolean

```
dbGetRowRegionUsesPartialRowSpec (  
    dbId rowRegionId  
);
```

Description

Confirms whether the fill space of rowRegion should be populated with partial specialization of rowRegionSpec.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.

Return Value

<i>Boolean</i>	TRUE if the fill space of rowRegion is populated with partial specialization of rowRegionSpec. FALSE if the fill space of rowRegion is not populated with partial specialization of rowRegionSpec.

dbIsRowRegion

Boolean

```
dbIsRowRegionSpec(  
    dbId objId  
);
```

Description

Checks if the specified database object is a rowRegion.

Arguments

<i>objId</i>	Database ID of an object.

Return Values

<i>Boolean</i>	TRUE if the object is a rowRegion. FALSE if the object is not a rowRegion.

dbIsRowRegionAttachedToPRBoundary

Boolean

```
dbIsRowRegionAttachedToPRBoundary(  
    dbId rowRegionId  
);
```

Description

Checks if the specified rowRegion is attached to PRBoundary.

Arguments

rowRegionId	Database ID of a rowRegion.

Return Values

<i>Boolean</i>	TRUE if the rowRegion is attached to PRBoundary. FALSE if the rowRegion is not attached to PRBoundary.

dbRebuildRowRegion

Boolean

```
dbRebuildRowRegion(  
    dbId rowRegionId  
);
```

Description

Removes contents of the specified rowRegion and then regenerates it based on the rowRegionSpec associated with it.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.

Return Values

<i>Boolean</i>	TRUE if the operation is successful. FALSE if the operation fails.

dbRowRegionMatchesSpec

Boolean

```
dbRowRegionMatchesSpec(  
    dbId rowRegionId  
);
```

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>rowRegionId</i>	Database ID of a rowRegion.

Return Values

<i>Boolean</i>	TRUE if the specified rowRegion matches its rowRegionSpec. FALSE if specified rowRegion does not match its rowRegionSpec.

dbSetRowRegionPoints

Boolean

```
dbSetRowRegionPoints(  
    dbId          rowRegionId,  
    unsigned int numPoints,  
    dbPoint       *pointArray  
);
```

Description

Changes the rectilinear specification of the given rowRegion.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.
<i>numPoints</i>	Number of points in pointArray.
<i>*pointArray</i>	Pointer to the point array.

Return Values

<i>Boolean</i>	TRUE if the rowRegion points are modified. FALSE if the operation fails.

dbSetRowRegionRowRegionSpec

Boolean

```
dbSetRowRegionRowRegionSpec (
    dbId          rowRegionId,
    dbRowRegionSpecId rowRegionSpecId
);
```

Description

Sets the given rowRegionSpec as the template for the specified rowRegion.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.
<i>rowRegionSpecId</i>	Database ID of a rowRegionSpec.

Return Values

<i>Boolean</i>	TRUE if the given rowRegionSpec is set as the row template for the specified rowRegion. FALSE if the operation fails.

dbSetRowRegionUsesPartialRowSpec

Boolean

```
dbSetRowRegionUsesPartialRowSpec (  
    dbId      rowRegionId,  
    Boolean value  
);
```

Description

A Boolean indicating whether the population of rowRegion is set to fill the space with partial specialization of rowRegionSpec.

Arguments

rowRegionId

Database ID of a rowRegion.

value

A Boolean that determines whether rowRegion should be set to use partial specialization of rowRegionSpec.

Return Values

Boolean

TRUE if the partial specialization of rowRegionSpec is set or unset.

FALSE if the operation fails.

dbStartGenRowRegionPlaceRows

```
dbGenStateId  
dbStartGenRowRegionPlaceRows (  
    dbRowRegionId  rowRegionId  
);
```

Description

Initializes a generator for all placement rows scoped to the given rowRegion object. A generator state ID is returned, which is used with the dbStartGenRowRegionPlaceRows function.

Arguments

<i>rowRegionId</i>	Database ID of a rowRegion.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbStartGenRowRegions

```
dbGenStateId  
dbStartGenRowRegions(  
    dbCellViewId cellViewId  
);
```

Description

Initializes a generator for all rowRegion objects. A generator state ID is returned, which is used with the [dbGenRowRegions](#) function.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Miscellaneous Functions

- [dbGetPlaceRowSoftwareVersion](#)
- [dbGetPlaceRowDataVersion](#)
- [dbGetRowRegionDataVersion](#)
- [dbGetRowRegionSoftwareVersion](#)
- [dbIsImplicit](#)
- [dbIsInRowRegion](#)

dbGetPlaceRowSoftwareVersion

```
unsigned int  
dbGetPlaceRowSoftwareVersion(  
    );
```

Description

Returns the software version of the placement row infrastructure in the given design.

Arguments

None

Return Value

<i>unsigned int</i>	Software version of the placement row infrastructure.

dbGetPlaceRowDataVersion

```
unsigned int  
dbGetPlaceRowDataVersion(  
    dbCellViewId cellViewId  
);
```

Description

Returns the data version of the placement row infrastructure in the given design in the specified cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>unsigned int</i>	Data version of the placement row infrastructure used in the specified cellview.

dbGetRowRegionDataVersion

`unsigned int`

```
dbGetRowRegionDataVersion(  
    dbCellViewId cellViewId  
);
```

Description

Returns the data version of the row region infrastructure in the given design in the specified cellview.

Arguments

<i>cellViewId</i>	Database ID of a cellview.

Return Value

<i>unsigned int</i>	Data version of the row region infrastructure used in the specified cellview.

dbGetRowRegionSoftwareVersion

```
unsigned int  
dbGetRowRegionSoftwareVersion(  
    );
```

Description

Returns the software version of the row region in the given design.

Arguments

None

Return Value

<i>unsigned int</i>	Version of the row region software.

dbIsImplicit

Boolean

```
dbIsImplicit(  
    dbId objId  
);
```

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>objId</i>	Database ID of the object to be queried.

Return Value

<i>Boolean</i>	TRUE if the object is implicit. FALSE if the object is not implicit.

dbIsInRowRegion

Boolean

```
dbIsInRowRegion(  
    dbFigId figId  
);
```

Description

Checks if a figure with the given figure database ID belongs to a rowRegion.

Arguments

<i>figId</i>	Database ID of a figure.

Return Value

<i>Boolean</i>	TRUE if the figure exists in or belongs to a rowRegion. FALSE if the figure does not belong to a rowRegion.

Cadence Integrators Toolkit Database Reference

CDBA Placement Functions

CDBA Parent-Child Relationship Functions

The parent-child relationship allows database objects in the same cellview to be tightly coupled together. The tight coupling lets the database support propagation of operations (such as move, copy, and delete) that, when acting on a parent, act on all the parent's descendants ("children") as well.

The `MatchPointLists` relationship can exist between a parent and its descendants. When this relationship exists, the parent's graphical points are the same as all the points of its descendants. If the parent has its points modified, then the descendants' points are modified too. Also, a matching child cannot be operated on (moved, copied, or deleted) by itself. The matching ancestor must be operated on instead.

- [Limitations](#) on page 982
- [Retrieval and Traversal Operations](#) on page 982
- [Predicates](#) on page 983
- [Creation and Deletion Operations](#) on page 984
- [Other CDBA Functions Affected](#) on page 986

Limitations

The following sections describe limitations to the parent-child relationship.

Allowed Database Objects

Currently, only figures (except mosaics) can be children. Parents are limited to figures (shapes, instances, and mosaics).

Uniqueness

A parent can have any number of children. However, a child can have only one parent.

Grandchildren/Descendants

Children can themselves be parents, allowing for *n* levels of hierarchy in the parent-child relationship. No cycles are allowed in the parent-child hierarchy.

Matching Points

The `MatchPointLists` constraint can be applied to two-point shapes (that is rectangles, implying that all descendants have to be rectangles) and multiple-point shapes such as path, line, and polygon. (Descendants can be any of path, line, or polygon to be compatible.)

Retrieval and Traversal Operations

The following functions traverse the hierarchy to retrieve and change information about parent-child relationships for a figure.

dbStartGenChild

```
dbGenStateId  
dbStartGenChild(  
    dbId parentId  
);
```

Starts the generator for all immediate children of a parent.

dbGenChild

Boolean

```
dbGenChild(  
    dbGenStateId stateId, dbId pChildId  
);
```

Generates all immediate children of a parent. When no more children exist for the parent, the functions return FALSE.

dbGetParent

dbId

```
dbGetParent(  
    dbId childId  
);
```

Returns the parent of the child object specified by `childId`. If the object specified by `childId` does not have a parent, the function returns `dbcNullId`.

Predicates

The following functions retrieve predicate (Boolean) information about parent-child relationships.

dbIsParent

Boolean

```
dbIsParent(  
    dbId parentId  
);
```

Returns TRUE if the object specified by `parentId` is a parent of one or more children. Otherwise, it returns FALSE.

dbIsChild

Boolean

```
dbIsChild(  
    dbId childId  
);
```

Cadence Integrators Toolkit Database Reference

CDBA Parent-Child Relationship Functions

Returns TRUE if the object specified by `childId` is the child of some parent. Otherwise, it returns FALSE.

dbIsChildOfParent

Boolean

```
dbIsChildOfParent(  
    dbId childId, dbId parentId  
);
```

Returns TRUE if the object specified by `childId` is a child of the object specified by `parentId`. Otherwise, the returns FALSE.

dbIsParentOfChild

Boolean

```
dbIsParentOfChild(  
    dbId parentId, dbId childId  
);
```

Returns TRUE if the object specified by `parentId` is the parent of the object specified by `childId`. Otherwise, the function returns FALSE.

dbHasMatchPointLists

Boolean

```
dbHasMatchPointLists(  
    dbId parentId  
);
```

Returns TRUE if the object specified by `parentId` has the `MatchPointLists` constraint. Otherwise, the function returns FALSE.

Creation and Deletion Operations

The following functions retrieve and change information about parent-child relationships a figure.

dbAddChildToParent

Boolean

Cadence Integrators Toolkit Database Reference

CDBA Parent-Child Relationship Functions

```
dbAddChildToParent(  
    dbId childId, dbId parentId  
);
```

Adds a parent-child relationship to a database object specified by `childId`, which is the child of the object specified by `parentId`. FALSE is returned if the relationship cannot be built.

dbSubChildFromParent

Boolean

```
dbSubChildFromParent(  
    dbId childId, dbId parentId  
);
```

Removes a child relationship from the object specified by `parentId` to the object specified by `childId`. FALSE is returned if there are problems removing the relationship.

dbMatchPointLists

Boolean

```
dbMatchPointLists(  
    dbId parentId  
);
```

Checks all the descendants of a parent. If all the points match, the function adds the `MatchPointLists` constraint to the parent and all its descendants. If successful, the function returns TRUE. Otherwise, the function returns FALSE. Any children added to any parent with the `MatchPointLists` constraint must match points before they can be added.

An application might require that all the points of a parent figure, specified by `parentId`, have the same points as all the descendants of that parent. This means that the number of points for the descendants must be the same as the parent, in addition to the coordinates themselves matching.

dbUnmatchPointLists

Boolean

```
dbUnmatchPointLists(  
    dbId parentId  
);
```

Removes the `MatchPointLists` constraint (that is, all descendants of a parent specified by `parentId` must match the parent's points). The routine returns TRUE if the constraint is removed properly. Otherwise, the function returns FALSE.

Other CDBA Functions Affected

See the descriptions of the following functions for more information regarding parent-child relationships:

- [dbStartGenFig and dbGenFig on page 210](#)
- [dbStartGenNonChildFig and dbGenNonChildFig on page 210](#)
- [dbStartGenLPToShape on page 412](#)
- [dbGenLPToShape on page 412](#)
- [dbStartGenLPToNonChildShape on page 413](#)
- [dbGenLPToNonChildShape on page 413](#)
- [dbStartGenShape on page 411](#)
- [dbGenShape on page 411](#)
- [dbStartGenNonChildShape on page 411](#)
- [dbGenNonChildShape on page 411](#)
- [dbCopyFig on page 212](#)
- [dbCopyShape on page 414](#)
- [dbMoveFig on page 211](#)
- [dbMoveShape on page 413](#)
- [dbDeleteFig on page 211](#)
- [dbDeleteShape on page 413](#)
- [dbDeleteInst on page 482](#)
- [dbDeleteMosaic on page 578](#)
- [dbSetPolygonPoint on page 420](#)
- [dbSetPathPoint on page 426](#)
- [dbSetLinePoint on page 423](#)
- [dbSetRectBBox on page 418](#)
- [dbConvertRectToPolygon on page 419](#)
- [dbConvertPolygonToRect on page 422](#)

Cadence Integrators Toolkit Database Reference

CDBA Parent-Child Relationship Functions

- [dbConvertPathToPolygon](#) on page 430

Cadence Integrators Toolkit Database Reference

CDBA Parent-Child Relationship Functions

CDBA Inherited Connection Functions

Inherited connections provide an extension to the connectivity model. Inherited connections allow you to create global signals and to override the names for global signals for selected branches of the design hierarchy. The flexibility of inherited connections allows you to

- Use multiple power supplies in a design
- Override substrate connections
- Use parameterized power and ground symbols

For a description of connectivity and naming conventions for inherited connections in Composer, and for a description of how to add and edit net expressions in a design, see the [Virtuoso Schematic Editor L User Guide](#).

This chapter gives you database API functions for inherited connections (presented alphabetically within each section), and examples for some of these functions. These functions cover the following general relationships:

- [Signals and Inherited Net Expressions](#) on page 990
- [Terminals and Inherited Net Expressions](#) on page 1003

Signals and Inherited Net Expressions

The following functions primarily involve the relationship between signals and inherited net expressions.

- dbCreateSigNetExpr
- dbCVHasInheritedSig
- dbDeleteCVInheritedSigs
- dbDeleteSigNetExpr
- dbGenInheritedSig
- dbGetSigNetExpr
- dbIsSigInherited
- dbIsValidSigNetExpr
- dbParseSigNetExpr
- dbReplaceSigNetExpr
- dbStartGenInheritedSig

dbCreateSigNetExpr

Boolean

```
dbCreateSigNetExpr(  
    dbSigId sigId,  
    String netExpression  
);
```

Description

Given an inherited net expression in `netExpression` and a signal identifier in `sigId`, this function attaches the inherited net expression to the given signal.

This function checks the syntax of the given net expression, but it does not detect whether the given default net name is global or whether the evaluation of the property results in a valid scalar net name.

This function also checks whether the inherited net expressions on the given signal and the formal terminal signal is connected to match or not.

Arguments

<code>sigId</code>	The database identifier of the signal.
<code>netExpression</code>	The inherited net expression.

Return Values

Boolean	TRUE is returned if the inherited net expression is successfully attached to the signal. FALSE is returned if the given signal does not exist, the given net expression has illegal syntax, or if the signal already has an inherited net expression associated with it.

dbCVHasInheritedSig

Boolean

```
dbCVHasInheritedSig(  
    dbCellViewId cellViewId  
);
```

Description

Given a cellview identifier in `cvId`, this function checks if there is any inheriting signal in the given cellview.

Arguments

<code>cellViewId</code>	The identifier of a given cellview which is checked for inheriting signals.

Return Values

Boolean	TRUE is returned if there is any inheriting signal in the given cellview. FALSE is returned if there is none or if the given cellview cannot be accessed.

dbDeleteCVInheritedSigs

Boolean

```
dbDeleteCVInheritedSigs(  
    dbCellViewId cellViewId  
);
```

Description

Given a cellview identifier in `cellViewId`, this function deletes inherited net expressions from all inherited signals in the given cellview.

Arguments

<code>cellViewId</code>	The identifier of a given cellview in which signals and their inherited net expressions are deleted.

Return Values

Boolean	TRUE is returned if all inherited signals have been deleted from the given cellview. FALSE is returned if the given cellview cannot be accessed; if there are no inherited signals in the given cellview; if any of the inherited net expressions associated with the inherited signals are not removed successfully.

dbDeleteSigNetExpr

Boolean

```
dbDeleteSigNetExpr(  
    dbSigId sigId  
);
```

Description

Given a signal identifier in `sigId`, this function deletes any existing inherited net expression associated with this signal.

Arguments

<code>sigId</code>	The identifier of the signal to which an inherited net expression is associated.

Return Values

Boolean	TRUE is returned if the inherited net expression exists on the object and is successfully deleted. FALSE is returned if no inherited net expression belongs to the object or if the given signal cannot be found.

dbGenInheritedSig

Boolean

```
dbGenInheritedSig(  
    dbGenStateId pState,  
    dbSigId sigId,  
);
```

Description

This function returns the identifier of a signal whose connection is inherited.

Cadence Integrators Toolkit Database Reference

CDBA Inherited Connection Functions

Arguments

pState	The state indicating whether there are inherited signals to check.
sigId	The identifier of a signal whose connection is inherited.

Return Values

Boolean	TRUE is returned if the generator functions correctly. FALSE is returned when there are no more inherited signals.

dbGetSigNetExpr

```
String  
dbGetSigNetExpr(  
    dbSigId sigId,  
    );
```

Description

Retrieves the inherited net expression associated with the signal.

Arguments

<code>sigId</code>	The identifier of the signal used to retrieve the inherited net expression associated with the signal.

Return Values

<code>String</code>	The inherited net expression, if found, is returned. NULL is returned if none is found or if the given signal cannot be found. The inherited net expression is stored as a permanent string and should not be freed by the caller.

dbIsSigInherited

Boolean

```
dbIsSigInherited(  
    dbSigId sigId  
);
```

Description

Given a signal identifier in `sigId`, this function checks if there is an inherited net expression attached to it.

Arguments

<code>sigId</code>	The identifier of the signal which is checked for an inherited net expression.

Return Values

Boolean	TRUE is returned if an inherited net expression is found to be attached to the given signal. FALSE is returned if the given signal cannot be found or if there is no inherited net expression attached to the given signal.

dbIsValidSigNetExpr

Boolean

```
dbIsValidSigNetExpr(  
    String netExpression  
);
```

Description

Given an inherited net expression in `netExpression`, this function checks whether its syntax is valid or not. 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

<code>netExpression</code>	The inherited net expression checked for syntax.

Return Values

Boolean	The function returns TRUE if the syntax is valid. Otherwise, it returns FALSE.

dbParseSigNetExpr

Boolean

```
dbParseSigNetExpr(  
    String inheritedNetExpression, dbPropScope *searchScope, String *propName,  
    String *formatString, String *defaultNetName  
);
```

Description

Given an inherited net expression, this function parses the expression to return the individual components of the expression which include the search scope, the lookup property name, the format string and the default net name.

Cadence Integrators Toolkit Database Reference

CDBA Inherited Connection Functions

Arguments

<code>inheritedNetExpression</code>	The inherited net expression that is parsed.
<code>*searchScope</code>	The search scope of the inherited net expression.
<code>*propName</code>	The lookup property name of the inherited net expression.
<code>*formatString</code>	The format string of the inherited net expression.
<code>*defaultNetName</code>	The default net name of the inherited net expression.

Return Values

<code>Boolean</code>	The name of the property to be searched is returned in <code>propName</code>; the format string in <code>formatString</code>, the search scope in <code>searchScope</code>, and the default net name in <code>defaultNetName</code>. <code>FALSE</code> is returned if the syntax of the given inherited net expression is illegal. <code>propName</code>, <code>formatString</code> and <code>defaultNetName</code> are stored as permanent strings and should not be freed by the calling function.

Example

Given the net expression found in the *schematic* view of the cell *inv*, [`@vdd:%:vdd!`], this function returns:

```
searchScope    dbcHierarchy
propName       "vdd"
formatString    "%";
defaultNetName "vdd!"
```


dbReplaceSigNetExpr

Boolean

```
dbReplaceSigNetExpr(  
    dbSigId sigId, String netExpression  
);
```

Description

Given a signal identifier in `sigId`, this function replaces any existing inherited net expression already associated with the signal. If none exists, the given `netExpression` is attached to the given signal of `sigId`.

This function checks the syntax of the given net expression, but it does not detect whether the given default net name is global or whether the evaluation of the property results in a valid scalar net name. This function also checks whether the inherited net expressions on the given signal and the formal terminal signal is connected to match or not.

Arguments

<code>sigId</code>	The identifier of the signal to which an inherited net expression is associated.
<code>netExpression</code>	The inherited net expression to replace an existing expression, if one is associated with a terminal, or attached to the specified terminal, if no expression is associated with the terminal.

Return Values

Boolean	TRUE is returned if either an existing inherited net expression on the object is successfully replaced, or if there is not any existing inherited net expression on the object and the given inherited net expression is successfully attached to the signal. FALSE is returned if the given signal does not exist or the given net expression is of illegal syntax.

dbStartGenInheritedSig

```
dbGenStateId  
dbStartGenInheritedSig(  
    dbCellViewId cellViewId  
);
```

Description

This function starts up the generator for all signals which have inherited net expressions attached to them in the given cellview.

Arguments

<code>cellViewId</code>	The identifier of the cellview containing signals with inherited net expressions attached.

Return Values

<code>dbGenStateId</code>	The identifier of the newly started-up generator is returned. NULL is returned if there are no inherited signals in the given cellview.

Terminals and Inherited Net Expressions

The following functions primarily involve the relationship between terminals and inherited net expressions.

- dbCreateTermNetExpr
- dbCVHasInherited Term
- dbDeleteCVInheritedTerms
- dbDeleteTermNetExpr
- dbGenInheritedTerm
- dbGetTermNetExpr
- dbIsTermInherited
- dbIsValidTermNetExpr
- dbParseTermNetExpr
- dbReplaceTermNetExpr
- dbStartGenInheritedTerm

dbCreateTermNetExpr

Boolean

```
dbCreateTermNetExpr(  
    dbTermId termId,  
    String netExpression  
);
```

Description

Given an inherited net expression in `netExpression` and a terminal identifier in `termId`, this function attaches the inherited net expression to the given terminal.

This function checks the syntax of the given net expression, but it does not detect whether the given default net name is global or whether the evaluation of the property results in a valid scalar net name.

This function also checks whether the inherited net expressions on the given terminal and the formal terminal signal is connected to match or not.

Arguments

<code>termId</code>	The database identifier of the terminal.
<code>netExpression</code>	The inherited net expression.

Return Values

Boolean	TRUE is returned if the inherited net expression is successfully attached to the terminal. FALSE is returned if the given terminal does not exist, the given net expression has illegal syntax, or if the terminal already has an inherited net expression associated with it.

dbCVHasInherited Term

Boolean

```
dbCVHasInheritedTerm(  
    dbCellViewId cellViewId  
);
```

Description

Given a cellview identifier in `cellViewId`, this function checks whether there is any inherited terminal in the given cellview.

Arguments

<code>cellViewId</code>	The identifier of a specific cellview.

Return Values

Boolean	TRUE is returned if there are inherited terminals in the given cellview. FALSE is returned if there is none or if the given cellview cannot be accessed.

dbDeleteCVInheritedTerms

Boolean

```
dbDeleteCVInheritedTerms(  
    dbCellViewId cellViewId  
);
```

Description

Given a cellview identifier in `cellViewId`, this function deletes inherited net expressions from all inherited terminals in the given cellview.

Arguments

<code>cellViewId</code>	The identifier of a specific cellview.

Return Values

Boolean	TRUE is returned if all inherited terminals have been deleted from the given cellview. FALSE is returned if the given cellview cannot be accessed; if there are no inherited terminals in the given cellview; if any of the inherited net expressions associated with the inherited terminals.

dbDeleteTermNetExpr

Boolean

```
dbDeleteTermNetExpr(  
    dbTermId termId  
);
```

Description

This function removes from the given terminal designated by `termId` the inherited net expression associated with it. Once removed, the given terminal is no longer an inherited terminal.

Cadence Integrators Toolkit Database Reference

CDBA Inherited Connection Functions

Arguments

<code>termId</code>	The identifier of the terminal associated with the inherited net expression.

Return Values

<code>Boolean</code>	TRUE is returned if the inherited net expression is successfully removed from the given terminal. FALSE is returned if the given terminal does not exist or if no net expression is found associated with the given terminal.

dbGenInheritedTerm

Boolean

```
dbGenInheritedTerm(  
    dbGenStateId pState, dbTermId *termId  
);
```

Description

This function returns the identifier of a terminal whose connection is inherited in the current cellview.

Arguments

<code>pState</code>	The indication of whether an inherited terminal is identified.
<code>*termId</code>	The identifier of a terminal whose connection is inherited in the current cellview.

Return Values

Boolean	The identifier of a terminal whose connection is inherited is returned in <code>termId</code> . TRUE is returned if the generator functions correctly. FALSE is returned when there are no more inherited terminals.

dbGetTermNetExpr

String

```
dbGetTermNetExpr(  
    dbTermId termId  
);
```

Description

Given a terminal identifier in `termId`, this function retrieves the inherited net expression associated with the given terminal.

Arguments

<code>termId</code>	The identifier of the terminal associated with the inherited connection.

Return Values

String	The inherited net expression, if found, is returned. NULL is returned if none is found. The inherited net expression is stored as a permanent string and should not be freed by the calling function.

dbIsTermInherited

Boolean

```
dbIsTermInherited(  
    dbTermId termId  
);
```

Description

Given a terminal identifier in `termId`, this function checks whether there is an inherited net expression attached to the given terminal.

Arguments

<code>termId</code>	The identifier of the terminal being checked for an inherited net expression attached.

Return Values

Boolean	TRUE is returned if an inherited net expression is found to be associated with the given terminal. FALSE is returned on error, indicating that the given terminal does not exist, or that the given terminal is not an inherited terminal.

dbIsValidTermNetExpr

Boolean

```
dbIsValidTermNetExpr(  
    String netExpression  
);
```

Description

Given an inherited net expression in `netExpression` associated with a terminal, this function checks whether its syntax is valid or not. This function does as much local checking as possible, but it does not detect whether the given default net name is global or whether the evaluation of the property results in a valid net name.

Arguments

<code>netExpression</code>	The inherited net expression checked for valid syntax.

Return Values

Boolean	TRUE is returned if the syntax of the given implicit terminal expression is valid; otherwise, FALSE is returned.

Example

[@bulk:%:gnd!] is legal syntax.

[@bulk:%] is illegal since the required default net name is missing.

[&bulk:%:bulk!] is illegal because the & character is not a valid search scope designator.

dbParseTermNetExpr

Boolean

```
dbParseTermNetExpr(  
    String inheritedNetExpression, dbPropScope *searchScope, String *propName,  
    String *formatString, String *defaultNetName  
);
```

Description

Given an inherited net expression attached to a terminal, this function parses the expression to return the individual components of the expression which include the search scope, the lookup property name, the format string and the default net name.

The `enum` in CDBA for the search scope is:

```
"@"      dbcHierarchy
```

Cadence Integrators Toolkit Database Reference

CDBA Inherited Connection Functions

Arguments

<code>inheritedNetExpression</code>	The inherited net expression attached to a terminal.
<code>*searchscope</code>	The search scope of the inherited net expression.
<code>*propName</code>	The lookup property name of the inherited net expression.
<code>*formatString</code>	The format string of the inherited net expression.
<code>*defaultNetName</code>	The default net name of the inherited net expression.

Return Values

<code>Boolean</code>	The name of the property to be searched is returned in <code>propName</code> ; the format string in <code>formatString</code> , the search scope in <code>searchScope</code> , and the default net name in <code>defaultNetName</code> . FALSE is returned if the given inherited net expression is of illegal syntax. <code>propName</code> , <code>formatString</code> and <code>defaultNetName</code> are stored as permanent strings and should not be freed by the caller.

Example

Given the net expression found in the *schematic* view of the cell *inv*, [`@vdd:%:vdd!`], this function returns:

```
searchScope      dbcHierarchy
propName         "vdd"
formatString     "%";
defaultNetName   "vdd!"
```

In the case where there are prefix and suffix in the format string, as in [`@gnd:my%:gnd!`], this function returns:

```
searchScope      dbcHierarchy
propName         "gnd"
formatString     "%";
defaultNetName   "gnd!"
```

dbReplaceTermNetExpr

Boolean

```
dbReplaceTermNetExpr(  
    dbTermId termId, String netExpression  
);
```

Description

Given a terminal identifier in `termId`, this function replaces any existing inherited net expression already associated with the terminal. If none exists, the given `netExpression` is attached to the given terminal.

This function does as much local checking as possible on the syntax of the given inherited net expression, but it does not detect whether the given default net name is global nor whether the evaluation of the property results in a valid scalar net name.

This function also checks whether the inherited net expressions on the formal terminal and the formal terminal signal is connected to match or not.

Arguments

<code>termId</code>	The terminal identifier to be associated with the net expression (by replacement or attachment).
<code>netExpression</code>	The inherited net expression to replace an existing expression, if one is associated with a terminal, or attached to the specified terminal, if no expression is associated with the terminal.

Return Values

Boolean	TRUE is returned under the following two scenarios: if an existing inherited net expression on the terminal is successfully replaced, or if there is not any existing inherited net expression on the object and the given inherited net expression is successfully attached to the terminal. FALSE is returned if there is an error, such as if a <code>termId</code> is bad, if the syntax of a given net expression is illegal, or if the inherited net expressions on the formal terminal and the formal terminal signal is connected to not match.

dbStartGenInheritedTerm

```
dbGenStateId  
dbStartGenInheritedTerm(  
    dbCellViewId cellViewId  
);
```

Description

Given a cellview identifier in `cellViewId`, this function starts up the generator for all terminals which have inherited net expressions associated with them.

Arguments

<code>cellViewId</code>	The identifier for a cellview that is being checked for inherited terminals.

Return Values

<code>dbGenStateId</code>	The identifier of the newly started-up generator is returned. NULL is returned if there is no inherited terminal in the given cellview.

Cadence Integrators Toolkit Database Reference

CDBA Inherited Connection Functions

CDF Read and Write Access Functions

This chapter describes Component Description Format (CDF) C-level read and write access functions for simulation applications. Read-access functions include query, CDF data retrieval, and generator functions. Write-access functions include those that create, delete, copy, update, and save data.

For more information on the use model for Component Description Format and on the CDF SKILL functions that work with the C functions described in this chapter, see the *Component Description Format User Guide*.

This chapter covers the following topics:

- [CDF Read-Access Functions](#) on page 1018
 - ❑ [Query Functions](#) on page 1019
 - ❑ [CDF Data Retrieval Functions](#) on page 1025
 - ❑ [Generator Functions](#) on page 1028
- [CDF Write-Access Functions](#) on page 1031
 - ❑ [Functions to Create Data](#) on page 1031
 - ❑ [Functions to Delete Data](#) on page 1042
 - ❑ [Functions to Copy Data](#) on page 1044
 - ❑ [Functions to Update Data](#) on page 1046
 - ❑ [The Function to Save Data](#) on page 1049

CDF Read-Access Functions

You can use CDF read-access functions to do the following:

- Query CDF descriptions for information.
- Access and retrieve CDF parameters and information.
- Access CDF information, including simulation and label information.

Cadence® digital applications use CDF descriptions.

The data structures for CDF read-access functions are defined in the `ITKDB/include/itkDB.h` file. For an example of the use of CDF read-access functions, see the `itkDB/Sample/CDF/main.c` file.

Additional CDF write-access functions have been added to versions of ITKDB. For an example of the use of write-access functions, see the `itkDB/Sample/CDF/CDFmainCreate.c` file.

See the *Component Description Format User Guide* for more detailed information on CDF descriptions.

Query Functions

The following functions query existing CDF descriptions to determine if a cell or library has any base- or user-defined CDF. Use the returned CDF handle to find a specific parameter or find all the parameters. (A handle is the identifier of the CDF data object through which the CDF data is accessed.)

cdfGetLibCDF

```
cdfDataId  
cdfGetLibCDF(  
    ddId libId  
);
```

Description

Returns the effective CDF description attached to a library. If neither a base- nor user-level CDF description is defined, the function returns `cdfcNullDataId`. The resulting CDF description represents the overlay of the user-level CDF onto the base-level CDF.

Arguments

<code>libId</code>	The database identifier of the library.

Return Values

<code>cdfDataId</code>	The effective CDF description attached to a library. If the library does not exist, this function returns <code>cdfcNullDataId</code> .

Example

```
cdfDataId cdfData;  
ddId libId;  
cdfData = cdfGetLibCDF( libId);
```

cdfGetBaseLibCDF

```
cdfDataId  
cdfGetBaseLibCDF(  
    ddId libId  
);
```

Description

Returns the base-level CDF description attached to a library.

Arguments

<code>libId</code>	The database identifier of the library.

Return Values

<code>cdfDataId</code>	The base-level CDF description attached to a library. If the library does not exist, this function returns <code>cdfcNullDataId</code> .

Example

```
cdfDataId cdfData;  
ddId libId;  
cdfData = cdfGetBaseLibCDF( libId);
```

cdfGetCellCDF

```
cdfDataId  
cdfGetCellCDF(  
    ddId cellId  
);
```

Description

Returns the effective CDF description attached to a cell. The resulting CDF description represents the overlay of the user-level cell CDF onto the base-level cell CDF; the base-level cell CDF onto the user-level library CDF; and the user-level library CDF onto the base-level library CDF.

Arguments

<code>cellId</code>	The database identifier of the cell.

Return Values

<code>cdfDataId</code>	The effective CDF description attached to a cell. If the cell does not exist, or if neither a base- nor user-level CDF description is defined for the cell or its library, the function returns <code>cdfcNullDataId</code> .

Example

```
cdfDataId cdfData;  
ddId cellId;  
cdfData = cdfGetBaseCellCDF( cellId);
```

cdfGetBaseCellCDF

```
cdfDataId  
cdfGetBaseCellCDF(  
    ddId cellId  
);
```

Description

Returns the base-level CDF description attached to a cell.

Arguments

<code>cellId</code>	The database identifier of the specified cell.

Return Values

<code>cdfDataId</code>	The effective CDF description attached to a cell. If the cell does not exist, or if the base-level CDF description is not defined, this function returns <code>cdfcNullDataId</code> .

Example

```
cdfDataId cdfData;  
ddId cellId;  
cdfData = cdfGetBaseCellCDF( cellId);
```

cdfGetInstCDF

```
cdfDataId  
cdfGetInstCDF(  
    dbAnyInstId anyInstId  
);
```

Description

Returns the effective CDF description associated with an instance. The difference between the instance's effective CDF description and the cell's effective CDF description is that the values of any CDF parameters take into account values of the parameters stored on the instance.

Arguments

<code>anyInstId</code>	The database identifier of a specified instance.

Return Values

<code>cdfDataId</code>	The effective CDF description associated with an instance. If the instance does not exist, this function returns <code>cdfcNullDataId</code> .

Example

```
cdfDataId cdfData;  
dbAnyInstId anyInstId;  
cdfData = cdfGetInstCDF( anyInstId);
```

cdfGetCDFType

```
cdfDataType  
cdfGetCDFType(  
    cdfDataId dataId  
);
```

Description

Returns the data type of the identifier of a CDF description.

The following table lists possible CDF types and a description of each.

CDF Data Types	Description
<code>cdfcLibDataType</code>	Effective-level library CDF
<code>cdfcBaseLibDataType</code>	Base-level library CDF
<code>cdfcUserLibDataType</code>	User-level library CDF
<code>cdfcCellDataType</code>	Effective-level cell CDF
<code>cdfcBaseCellDataType</code>	Base-level cell CDF
<code>cdfcUserCellDataType</code>	User-level cell CDF
<code>cdfcInstDataType</code>	Effective instance CDF

Arguments

<code>dataId</code>	The identifier of a CDF description.

Return Values

<code>cdfDataType</code>	The data type of a particular identifier of a CDF description. If the <code>dataId</code> is not valid, <code>cdfcUnknownDataType</code> is returned.

Example

```
cdfDataType dataType;  
cdfDataId dataId;  
dataType = cdfGetCDFType( dataId );
```


CDF Data Retrieval Functions

The following functions retrieve CDF parameters and information.

cdfFindParamByName

```
cdfParamId  
cdfFindParamByName(  
    cdfDataId dataId, String name  
);
```

Description

Returns the parameter identifier attached to a specified name and identifier of a CDF description, if the CDF description exists.

Arguments

<code>dataId</code>	The identifier of a CDF description.
<code>name</code>	The parameter name to search for.

Return Values

<code>cdfParamId</code>	The parameter identifier. If the parameter does not exist, this function returns <code>cdfcNullParamId</code> .

Example

```
cdfParamId param;  
cdfDataId dataId;  
String name;  
param = cdfFindParamByName( dataId, name);
```

cdfGetParamRec

```
cdfParamRec*  
cdfGetParamRec(  
    cdfParamId paramId  
);
```

Description

Returns a record structure containing the CDF parameter information about the specified parameter. The record structure is static within this routine. Subsequent calls to this function overwrite old data. The returned parameter information includes parameter type, parameter value, and default value.

Arguments

<code>paramId</code>	The <code>paramId</code> is the parameter identifier returned by either <code>cdfFindParamByName</code> or <code>cdfGenParam</code> .

Return Values

<code>cdfParamRec</code>	The record structure containing the CDF parameter. If the function is unable to retrieve this record structure, it returns <code>cdfcNullParamRec</code> .

Example

```
cdfParamId paramId;  
cdfParamRec *paramRec;  
paramRec = cdfGetParamRec( paramId );
```

cdfGetDataRec

```
cdfDataRec*  
cdfGetDataRec(  
    cdfDataId dataId  
);
```

Description

Returns a record structure containing information about the CDF description. The information includes the simulation, label, and other CDF information, such as `doneProc`, `fieldWidth`, and `fieldHeight`. The simulation and label information is stored as a SKILL list. You need to use the IL accessing/traversing functions (such as `ilCar`) to convert the lists to C data types. The record structure is static within this routine. Subsequent calls to this function overwrite old data.

Arguments

<code>dataId</code>	The identifier of a CDF description.

Return Values

<code>cdfDataRec</code>	The record structure containing information about a CDF description. If the CDF description does not exist, this function returns <code>cdfcNullDataRec</code> .

Example

```
cdfDataRec *cdfData;  
cdfDataId dataId;  
cdfData = cdfGetDataRec( dataId);
```

Generator Functions

The following functions work together in the process of starting the generator, of generating, and of stopping the generator for parameters of a given CDF identifier.

cdfStartGenParam

```
cdfStateId  
cdfStartGenParam(  
    cdfDataId dataId  
);
```

Description

Initializes the generator for generating all parameters on the specified `dataId`. Returns the generator state identifier. You must call `cdfStopGenParam` to stop the generator.

Arguments

<code>dataId</code>	The identifier of a CDF description.

Return Values

<code>cdfStateId</code>	The generator state identifier. If the CDF description does not exist, this function returns <code>cdfcNullStateId</code> .

Example

```
cdfStateId state;  
cdfDataId dataId;  
state = cdfStartGenParam( dataId);
```

cdfGenParam

Boolean

```
cdfGenParam(  
    cdfStateId state, cdfParamId *pParamId  
);
```

Description

Generates the parameters of a CDF description, given the generator state identifier initialized with the call to `cdfStartGenParam`. The generator `state` is updated to reflect the new state. The `pParamID` returns a pointer to the parameter identifier.

Arguments

<code>state</code>	The generator state.
<code>pParamId</code>	Pointer to a generated parameter identifier, if the function returns TRUE.

Return Values

Boolean	The function returns FALSE if there are no more parameters. Otherwise, the function returns TRUE.

Example

```
cdfStateId state;  
cdfDataId dataId;  
cdfParamId paramId;  
state = cdfStartGenParam( dataId);  
while (cdfGenParam(state, & paramId))  
{/*process paramId*/}  
cdfStopGen(state);
```

The `/*process paramId*/` comment in the example indicates that you, the programmer are to specify what to do to process the `paramId`.

cdfStopGen

Boolean

```
cdfStopGen(  
    cdfStateId state  
);
```

Description

Stops the specified generator and invalidates the generator state structure at the end of the parameter generator loop.

Arguments

state	The identifier of the generator state.

Return Values

Boolean	If the state is not valid, this function returns FALSE. Otherwise, the function returns TRUE.

Example

```
cdfStateId state;  
cdfcDataId dataId;  
cdfParamId paramId;  
state = cdfStartGenParam( dataId);  
while (cdfGenParam(state, & paramId))  
{/*process paramId*/}  
cdfStopGen(state);
```

CDF Write-Access Functions

CDF write-access functions include those that create, delete, copy, update, or save data.

Functions to Create Data

The functions in this section create CDF information in the specified cell or library.

cdfCreateBaseCellCDF

```
cdfDataId  
cdfCreateBaseCellCDF(  
    ddId cellId, String doneProc, String formInitProc, int fieldWidth, int  
    fieldHeight, int buttonFieldWidth, int promptWidth  
);
```

Description

Creates and returns a new identifier for a base-level CDF description (`cdfDataId`) of the specified cell identifier (`cellId`). The cell must not already have a base-level CDF description defined.

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

Arguments

<code>cellId</code>	The database identifier of the cell.
<code>doneProc</code>	An optional SKILL procedure name (a string) that is evaluated after any change to a parameter on a component instance. You can use <code>doneProc</code> for postprocessing. The procedure must take a single argument which is a list that contains the instance that has been modified. For examples of the forms that use this argument, see Chapter 8 of the <i>Component Description Format User Guide</i>.
<code>formInitProc</code>	An optional SKILL procedure name that you specify. The procedure is executed when the contents of the CDF are displayed on a form such as the Edit Object Properties form. This procedure runs when you use the <i>Add Component</i> and <i>Edit Properties</i> menu commands. You can use the procedure for preprocessing CDF data. This procedure must take a single argument, the <code>cdfDataId</code> being added to the form. For examples of the forms that use this argument, see Chapter 8 of the <i>Component Description Format User Guide</i>. When you modify a parameter field value using the <code>formInitProc</code>, the <i>Edit Properties Object</i> command for the schematic application is not aware of the modification and does not update the changes made with <code>formInitProc</code>. You can avoid this problem by setting the variable <code>cdfgForm->cdfModified</code> to the value <code>t</code>.
<code>fieldWidth</code>	The width of the field on the instantiation form. The default is 350 pixels. For examples of the forms that use this argument, see Chapter 6 of the <i>Component Description Format User Guide</i> .
<code>fieldHeight</code>	The height of the field on the instantiation form. The default is 35 pixels. For examples of the forms that use this argument, see Chapter 6 of the <i>Component Description Format User Guide</i> .
<code>buttonFieldWidth</code>	The width of a button on the instantiation form. The default is 350 pixels. For examples of the forms that use this argument, see Chapter 6 of the <i>Component Description Format User Guide</i> .

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

`promptWidth` The width of the prompt window on the instantiation form. The default is 175 pixels. For examples of the forms that use this argument, see Chapter 6 of the *Component Description Format User Guide*.

Return Values

`cdfDataId` The identifier of the CDF description. If the description does not exist, this function returns `cdfcNullDataId`.

Example

```
cdfDataId cdfData;
ddId cellId;
String doneProc;
String formInitProc;
int fieldWidth, fieldHeight, buttonFieldWidth, promptWidth;
cdfData = cdfCreateBaseCellCDF( cellId, doneProc, formInitProc,
                                fieldWidth, fieldHeight,
                                buttonFieldWidth, promptWidth);
```

cdfCreateBaseLibCDF

```
cdfDataId  
cdfCreateBaseLibCDF(  
    ddId libId,  
    String doneProc,  
    String formInitProc,  
    int fieldWidth,  
    int fieldHeight,  
    int buttonFieldWidth,  
    int promptWidth  
);
```

Description

Creates and returns the base-level CDF description (`cdfDataId`) attached to a library (specified by `libId`). The library must not already have a base-level CDF description defined.

Arguments

<code>libId</code>	The database object (library identifier) to which the <code>cdfDataId</code> is attached.
<code>doneProc</code>	An optional SKILL procedure name (a string) that is evaluated after any change to a parameter on a component instance. You can use <code>doneProc</code> for postprocessing. The procedure must take a single argument which is a list that contains the instance that has been modified. For examples of the forms that use this argument, see Chapter 8 of the <i>Component Description Format User Guide</i>.
<code>formInitProc</code>	An optional SKILL procedure name. If specified, the procedure is executed when the contents of the CDF are displayed on a form. The default value is <code>cdfcNullDataId</code>. This procedure runs when you use the <i>Add Component</i> and <i>Edit Properties</i> commands. You can use the procedure for preprocessing CDF data. This procedure must take a single argument, the <code>cdfDataId</code> being added to the form. For examples of the forms that use this argument, see Chapter 8 of the <i>Component Description Format User Guide</i>.

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

When you modify a parameter field value using the `formInitProc`, the *Edit Properties Object* command for the schematic application is not aware of the modification and does not update the changes made with `formInitProc`. You can avoid this problem by setting the variable

`cdfgForm->cdfModified` to the value `t`.

fieldWidth The width of the field. The default is 350 pixels. For examples of the forms that use this argument, see Chapter 6 of the *Component Description Format User Guide*.

fieldHeight The height of the field. The default is 35 pixels. For examples of the forms that use this argument, see Chapter 6 of the *Component Description Format User Guide*.

buttonFieldWidth The width of the field of a button. The default is 340 pixels. For examples of the forms that use this argument, see Chapter 6 of the *Component Description Format User Guide*.

promptWidth The width of the prompt window. The default is 175 pixels. For examples of the forms that use this argument, see Chapter 6 of the *Component Description Format User Guide*.

Return Values

cdfDataId The base-level description attached to the specified library. If the library does not exist, this function returns `cdfcNullDataId`.

Example

The following example, creates base Lib CDF for analogLib library:

```
/*ddId of analogLib */
ddId analogLibId = ddGetObj("analogLib");

String doneProc = "formDoneProcedure()";
String formInitProc = "formInitProcedure()";

int fieldWidth = 350;
int fieldHeight = 35;
int buttonFieldWidth = 340;
int promptWidth = 175;
```

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

```
cdfDataId cdfData = cdfCreateBaseLibCDF(analogLibId, doneProc, formInitProc,  
fieldWidth, fieldHeight, buttonFieldWidth, promptWidth);
```

cdfCreateParam

```
cdfParamId  
cdfCreateParam(  
    cdfDataId dataId, String name, cdfParamType type, Boolean setDefValue,  
    cdfParamValue defValue, cdfUnitType units, cdfOptBoolean parseAsNumber,  
    unsigned int nChoices, String *choices, String prompt, String use, String  
    display, String editable, String dontSave, String callback, cdfOptBoolean  
    storeDefault, cdfOptBoolean parseAsCEL  
);
```

Description

Creates and returns a new parameter with the specified attributes attached to the specified `dataId`. The parameter has the specified `name`. The new parameter's name must not match an existing parameter's name on the same `dataId`.

If the `dataId` is a `baseCDF`, the user must have write permission on the object of the `baseCDF`. If the operation is successful, the new parameter's `cdfParamId` is returned.

Arguments

<i>dataId</i>	The identifier of the CDF description.
<i>name</i>	The name of the new parameter.
<i>type</i>	The data type of the new parameter.
<i>setDefValue</i>	Specifies whether to set the default value or not.
<i>defValue</i>	The default value for the new parameter.
<i>units</i>	The unit type of the parameter. You often modify parameters that represent values such as resistance, capacitance, length, time, or power, and expect to see the parameter value scaled to appropriate units (such as pF, uM, dBm). See the type <code>cdfUnitTypeEnums</code> in the <code>itkDB.h</code> file for a list of valid values for this argument. This argument determines the units suffix and scale factor to use when displaying the parameter value. For example, by setting a parameter's <code>units</code> field to capacitance, the parameter value is displayed as 5 pF instead of 5e-12.
<i>parseAsNumber</i>	If you specify this argument as <code>cdfcYes</code> , the system parses the value of a string parameter to check if the value is a number.

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

If the value is a number, the system converts the string to a floating point number, converts the string to the most efficient notation (taking into account any units field specified), then reconverts the number into a string.

If the parameter value is not a number, the system does no conversion.

For example, if the parameter value is the string *5.4e-12* and the parameter value has a units field of *capacitance* (that is, the suffix is F), the parameter value is converted to the string *5.4pF* before being displayed. If, however, the parameter value is *c1*, no conversion takes place.

nChoices

The number of choices.

**choices*

Possible values (a list of strings) of a *cyclic* or *radio* type parameter. For other types of parameters, specify as NULL and specify nChoices as 0.

prompt

Prompt that appears in a field on a form such as the Edit Object Properties form when an application asks for the value of this parameter.

use

Specifies whether to use this parameter. The *use* attribute is context-specific and is evaluated when necessary. In this field, you can specify parameters that you are not using because of the value of other parameters, or because of the function you are using.

The system evaluates the use field as a SKILL expression. If the argument is evaluated as non-nil, the system uses the parameter. If the argument is evaluated as nil, the system does not use the parameter. If you specify this argument as `dbcNullString` or `NULL`, it is assumed, so the system always uses the field.

See the “Global Variables” section in chapter 8 of the *Component Description Format User Guide* for a description of several global variables for constructing the *use* expression.

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

<i>display</i>	Determines if this parameter is displayed in forms that display CDF parameters, such as the Edit Properties form or the Add Component form. You must enter a string "t", "nil", or a SKILL expression that is evaluated as <i>t</i> or <i>nil</i> in this argument to determine if this parameter is to be displayed. If the argument is evaluated as non-<i>nil</i> (the default), the parameter is displayed. If the argument is evaluated as <i>nil</i>, the parameter is not displayed. If you specify this argument as <code>dbcNullString</code> or <code>NULL</code>, <i>t</i> is assumed and the system displays the field.
<i>editable</i>	the field specifies a displayed parameter that you cannot change. (Other parameters' callbacks can change the parameter.) Use this argument only on <i>int</i>, <i>float</i>, and <i>string</i> type parameters. The <i>editable</i> argument is evaluated as a SKILL expression. If the argument is evaluated as <i>non-nil</i>, the parameter is editable. out when displayed as a form field. If you specify this argument as <code>dbcNullString</code> or <code>NULL</code>, <i>t</i> is assumed, meaning that the field is editable.
<i>dontSave</i>	Specifies that the parameter cannot be stored in any instance that corresponds to the component. The system evaluates the argument as a SKILL expression. If the argument is evaluated as non-<i>nil</i>, the parameter is not stored. If you specify this argument as <code>dbcNullString</code> or <code>NULL</code>, <i>nil</i> is assumed and the parameter value is stored. This is the recommended usage. See Appendix A in the <i>Component Description Format User Guide</i> for additional usage advice.
<i>callback</i>	SKILL code to be executed whenever the value of a parameter changes. Using <i>callback</i>, you can cause any parameter's value to affect any other parameter's value. The value of this argument is a string that represents a SKILL expression or function. This argument is optional. If you specify the <i>callback</i> as <code>dbcNullString</code> or <code>NULL</code>, no callback is executed. See the "Global Variables" section in chapter 8 of the <i>Component Description Format User Guide</i> for a description of several global variables you can use in the <i>callback</i> function.

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

storeDefault

Specifies whether the parameter default value is saved as a property of the instance.

If set to `cdfcNo` (the default) or `cdfcUnsetBoolean`, a property is not saved on the instance when the parameter value is the default. Also, if the default value of the parameter changes, all instances that use the default automatically get the new default value. (To see the change in an open window, you must select *Window – Redraw* from the Cadence menu.)

If set to `cdfcYes` and the parameter value is set to the same value as the default, a property is saved on the instance. One disadvantage of this usage is that if the default value of a parameter changes, the default value will not change automatically on all the instances that have this parameter.

parseAsCEL

If set to `cdfcYes`, the associated CDF parameter is processed as a CDF Expression Language (CEL) expression. If the expression can be resolved to a numeric value (the usual case), the `parseAsNumber` flag should also be set to `yes`.

Return Value

cdfParamId

The identifier of the new parameter.

Example

The following example, creates an integer type parameter `length` for the cellview `analogLib/cap` attached to the Base CDF:

```
/* get cdfDataId of analogLib/cap */
ddId cellId = ddGetObj("analogLib", "cap");
cdfDataId dataId = cdfGetBaseCellCDF(cellId);

String name = "l";
cdfParamType type = cdfcIntParamType;
Boolean setDefValue = 1;
cdfParamValue defValue.aInt = 10;
cdfUnitType units = cdflenMetricUnit;
cdfOptBoolean parseAsNumber = cdfcYes;
String prompt = "length";
String use = NULL;
```


Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

```
String display = "t";
String editable = "t";
String dontSave = NULL;
String callback = "modelNameChanged()";
cdfOptBoolean storeDefault = cdfcNo;
cdfOptBoolean parseAsCEL = cdfcYes;
```

```
cdfParamId paramId = cdfCreateParam( dataId, name, type, setDefValue, defValue,
units, parseAsNumber, 0, 0, prompt, use, display, editable, dontSave, callback,
storeDefault, parseAsCEL);
```

Functions to Delete Data

The following functions delete identifier information from the specified cell or library.

cdfDeleteCDF

Boolean

```
cdfDeleteCDF(  
    cdfDataId dataId  
);
```

Description

Deletes the specified `cdfDataId` from the cell or library that owns it. If the `cdfDataId` is a base CDF, you must have write permission.

Arguments

<code>dataId</code>	The identifier of the CDF description.

Return Values

Boolean	If the data identifier is not valid, the function returns FALSE.

Example

```
cdfDataId dataId;  
cdfDeleteCDF( dataId);
```

cdfDeleteParam

Boolean

```
cdfDeleteParam(  
    cdfParamId paramId  
);
```

Description

Deletes the specified `cdfParamId` from the `cdfDataId` that owns it. If this parameter is from a `baseCDF`, the user must have write permission on the object of the `baseCDF`.

Arguments

<code>paramId</code>	The parameter to delete.

Return Values

Boolean	If the <code>cdfParamId</code> is successfully deleted, TRUE is returned. The function returns FALSE, if the parameter identifier is not valid.

Example

```
cdfParamId paramId;  
cdfDeleteParam( paramId);
```

Functions to Copy Data

The following functions copy identifiers to other identifiers.

cdfCopyCDF

```
cdfDataId  
cdfCopyCDF(  
    ddId id, cdfDataType type, cdfDataId dataId  
);
```

Description

Copies a CDF description to a specified identifier, changing the copy to be the specified type.

Arguments

<code>id</code>	Library or cell data identifier.
<code>type</code>	The data type of the CDF description to be copied.
<code>dataId</code>	The identifier of the CDF description to be copied.

Return Values

<code>cdfDataId</code>	The destination identifier for the description's copy. The function returns <code>cdfcNullDataId</code> if the copy operation is not successful.

Example

```
ddId id;  
cdfDataType type;  
cdfDataId dataId;  
cdfDataId copyId;  
copyId = cdfCopyCDF( id, type, dataId);
```

cdfCopyParam

```
cdfParamId  
cdfCopyParam(  
    cdfDataId dataId, cdfParamId paramId  
);
```

Description

Copies the specified parameter to the specified data CDF. If the data CDF identifier is a base-level CDF description (of type `baseCDF`), the user must have write permission on the object of the CDF. The specified parameter must not already exist in the destination CDF.

Arguments

<code>dataId</code>	The data identifier of the CDF description.
<code>paramId</code>	The parameter identifier of the CDF description.

Return Values

<code>cdfParamId</code>	If the copy is successful, the new <code>cdfParamId</code> is returned. Otherwise, <code>cdfcNullParamId</code> is returned.

Example

```
cdfParamId destParam;  
cdfDataId dataId;  
cdfParamId sourceParam;  
destParam = cdfCopyParam( dataId, sourceParam);
```

Functions to Update Data

The following functions update instances, CDFs, cells, or libraries, with the specified information.

cdfUpdateInstParam

Boolean

```
cdfUpdateInstParam(  
    dbAnyInstId anyInstId  
);
```

Description

Stores the parameters specified in the instance master's effective cell CDF onto the specified instance. The `doneProc` function is called if the instance exists.

Arguments

<code>anyInstId</code>	The database identifier for the instance.

Return Values

Boolean	Returns TRUE if the operation is successful.

Example

```
Boolean status;  
ddId anyInstId;  
status = cdfUpdateInstParam( anyInstId);
```

cdfUpdateInstSingleParam

Boolean

```
cdfUpdateInstSingleParam(  
    dbAnyInstId anyInstId, cdfParamId paramId  
);
```

Description

Stores the specified parameter in the instance master's effective cell CDF onto the specified instance.

Arguments

<code>anyInstId</code>	The database identifier of the instance.
<code>paramId</code>	The identifier of the parameter to be updated.

Return Values

Boolean	The function returns TRUE if the operation is successful. Otherwise, it returns FALSE.

Example

```
Boolean status;  
dbAnyInstId anyInstId;  
cdfParamId paramId;  
status = cdfUpdateInstSingleParam( anyInstId, paramId);
```

cdfSyncInstParamValue

Boolean

```
cdfSyncInstParamValue(  
    dbAnyInstId anyInstId1, dbAnyInstId anyInstId2  
);
```

Description

Generates all of the CDF parameters for an instance (`anyInstId1`) and updates a second instance (`anyInstId2`) to have the same values.

Arguments

<code>anyInstId1</code>	The parameter for which CDF parameters are being generated.
<code>anyInstId2</code>	The instance whose values are being updated.

Return Values

Boolean	The function returns TRUE if the operation is successful. Otherwise, the function returns FALSE.

Example

```
Boolean status;  
dbAnyInstId anyInstId1;  
dbAnyInstId anyInstId2;  
status = cdfSyncInstParamValue( anyInstId1, AnyInstId2);
```


The Function to Save Data

The following function saves data to the cell or library that owns it.

cdfSaveCDF

Boolean

```
cdfSaveCDF(  
    cdfDataId dataId  
);
```

Description

This function saves the specified CDF description (`cdfDataId`) to the cell or library that owns that CDF description.

Arguments

<code>dataId</code>	The identifier of the effective CDF description.

Return Values

Boolean	This function returns TRUE if the CDF description is successfully saved. Otherwise, it returns FALSE.

Example

```
Boolean status;  
cdfDataId dataId;  
status = cdfSaveCDF( dataId);
```

Cadence Integrators Toolkit Database Reference

CDF Read and Write Access Functions

CDBA Relative Object Design Functions

This chapter describes Relative Object Design (ROD) C database functions.

- For more information about ROD objects, refer to the *Virtuoso Relative Object Design User Guide*

Overview

Virtuoso relative object design (ROD) is functionality for defining simple and complex layout objects and their relationships to each other, without the need to use low-level Cadence SKILL language functions. ROD lets you create objects and define their relationships at a high level of abstraction, so you can concentrate on your design objectives. ROD automatically handles the intricacies of traversing the design hierarchy and simplifies the calculations required to create and align geometries.

With a single ROD function call, you can accomplish a task that otherwise would require several lower-level SKILL function calls. For example, creating a pin required a series of low-level SKILL function calls, but with ROD, you use a single function to create a shape and designate it as a pin. You can create entities such as guard rings, contact arrays, and transistors with one function call. You can also create an object from an existing object just by specifying the size of the new object.

About ROD Objects and ROD Object IDs

Every named database object, such as an instance, layout cellview, or named shape, automatically has relative object design information associated with it. This information is stored in a *ROD object*. A ROD object is also a database object, but it exists in relation to its associated named database object. A ROD object is identified by a unique *ROD object ID*.

A ROD object for a named shape, instance, or cellview contains the following information:

- hierarchical name
- cellview ID
- database ID
- transformation information (rotation, magnification, and offset)
- alignment information, if any
- number of segments (for shapes)
- names and values of user-defined handles, if any
- names of system-defined handles

A *ROD object ID* is similar to a database ID. The database ID for an object is the temporary address in memory for the object. A ROD object ID is the temporary address for the ROD object associated with a specific database object. When you look at these two IDs in the Command Interpreter Window (CIW) for a database object, the database ID might be `db:98342974` and the ROD object ID `rodObj:23970843`.

`cvId`

ID of the top-level cellview containing the ROD object.

<code>dbId</code>	Database ID of the ROD object at the lowest level. For multipart paths, <code>dbId</code> identifies the master path.

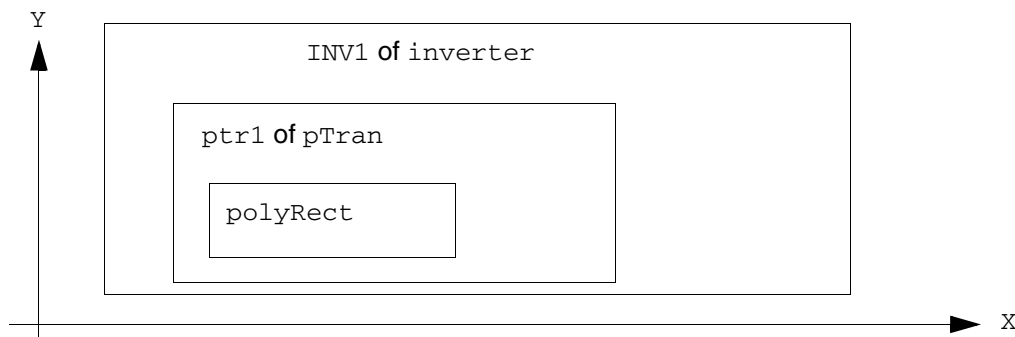
Named Objects

ROD lets you assign a name to a zero-level object (an ordinary database shape, such as a rectangle or polygon) either by naming an existing object or by using a ROD function to create an object. You can access information through hierarchy about named objects, such as instances and named shapes.

Hierarchical Name

To access information about a named shape or instance through hierarchy, use its *hierarchical name* and the top-level cellview ID. A hierarchical name consists of the names of the instances through which you need to descend to reach the desired named shape or instance.

The following figure shows the hierarchy in a layout cellview containing the named shape `polyRect`. Its hierarchical name is `INV1/ptr1/polyRect`.



When you name an existing database shape using the ROD naming function, the function creates ROD information associated with the shape. This information is stored in a *ROD object* and is identified by a unique ROD object ID. The information contained in a ROD object includes its name and database ID. When you create a new shape with a ROD creation function, the function creates a named database shape and a ROD object.

Boolean Query Functions

rodIsMPPMaster

Boolean

```
rodIsMPPMaster(  
    dbId          id  
);
```

Description

Determines if the specified *id* is the *dbId* for the master path of a ROD multipart path.

A multipart path (MPP) is a single ROD object consisting of one or more parts at level zero in the hierarchy on the same or on different layers.

A multipart path consists of a single *master path* and one or more *subparts*. The master path is an ordinary path; however, it is the defining part of a multipart path; all subparts are based on the master path. The subparts can be any combination of offset subpaths, enclosure subpaths, and sets of subrectangles.

Arguments

<i>id</i>	The database <i>id</i> for the master path of a ROD multipart path.

Return Values

Boolean	Returns TRUE if the specified <i>id</i> is the <i>dbId</i> for the master path of a ROD multipart path. Returns FALSE otherwise.

rodIsMPRMaster

Boolean

```
rodIsMPRMaster(  
    dbId          id  
);
```

Description

Determines if the specified *id* is the *dbId* for the master rectangle of a ROD multipart rectangle.

A multipart rectangle (MPR) is a single ROD object consisting of one or more parts at level zero in the hierarchy on the same or on different layers.

A multipart rectangle consists of a single *master rectangle* and one or more *subparts*. The master rectangle is an ordinary rectangle; however, it is the defining part of a multipart rectangle; all subparts are based on the master rectangle. The subparts can be any combination of offset subrectangles, enclosure subrectangles, and sets of subrectangles.

Arguments

<i>id</i>	The database <i>id</i> for the master rectangle of a ROD multipart rectangle.

Return Values

Boolean	Returns TRUE if the specified <i>id</i> is the <i>dbId</i> for the master rectangle of a ROD multipart rectangle. Returns FALSE otherwise.

rodIsMPPMasterChoppable

Boolean

```
rodIsMPPMasterChoppable(  
    dbId      id  
);
```

Description

Determines if the specified *id* is the *dbId* for the master path of a ROD multipart path and if that master path is choppable.

When the master path is choppable, then all of its subpaths and subrectangles must be choppable also. When the master path is not choppable, each subpath or set of subrectangles may be choppable or not choppable.

Arguments

<i>id</i>	The database <i>id</i> for the master path of a ROD multipart path.

Return Values

Boolean	Returns TRUE if the specified <i>id</i> is the <i>dbId</i> for the master path of a ROD multipart path and if that master path is choppable. Returns FALSE otherwise.

rodIsMPPSubShape

Boolean

```
rodIsMPPSubShape (
    dbId      id
);
```

Description

Determines if the specified *id* is a *dbId* for a subshape that is part of a ROD multipart path. The master path is not considered a subshape.

Arguments

<i>id</i>	The database <i>id</i> for the master path of a ROD multipart path.

Return Values

Boolean	Returns TRUE if the specified <i>id</i> is a <i>dbId</i> for a subshape that is part of a ROD multipart path. Returns FALSE otherwise. The master path is not considered a subshape; if the specified <i>id</i> is the <i>dbId</i> for the master path of a ROD multipart path, FALSE is returned.

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodIsObj

Boolean

```
rodIsObj (  
    rodObjectId      objId  
);
```

Description

Determines if the specified object *objId* is truly a ROD object.

Arguments

<i>objId</i>	ROD object id for the object tested.

Return Values

Boolean	Returns TRUE if the specified object <i>objId</i> is truly a ROD object. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodIsObjAligned

Boolean

```
rodIsObjAligned(  
    rodObjectId      objId  
);
```

Description

Determines if the specified ROD object *objId* is aligned.

Arguments

<i>objId</i>	ROD object <i>id</i> for the object tested.

Return Values

Boolean	Returns TRUE if the specified ROD object <i>objId</i> is aligned. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodIsObjShape

Boolean

```
rodIsObjShape (
    dbId      id
)
```

Description

Determines if the specified *id* is the database Id of a ROD object or is a subshape in a ROD multipart path or in a multipart rectangle.

Arguments

<i>id</i>	The <code>dbId</code> of a ROD object or a subshape in a ROD multipart path.

Return Values

Boolean	Returns TRUE if the specified <i>id</i> is the <code>dbId</code> of a ROD object or is a subshape in a ROD multipart path. Returns FALSE otherwise.

Object Query Functions

rodGetObjByDbId

```
rodObjectId  
rodGetObjByDbId(  
    dbId          id  
);
```

Description

Finds a ROD object specified by `dbId`.

The `dbId` is the database `id` of the ROD object at the lowest level. For multipart paths, `dbId` identifies the master path, for multipart rectangles it identifies the master rectangle.

Arguments

<i>id</i>	The <code>dbId</code> of a ROD object.

Return Values

<code>rodObjectId</code>	ROD object <i>id</i> for the object specified by <code>dbId</code> .

rodGetObjByName

```
rodObjectId  
rodGetObjByName (  
    String      hierName,  
    dbCellViewId cvId  
);
```

Description

Finds a ROD object *id* specified by hierarchical name.

Arguments

<i>hierName</i>	The hierarchical name of a ROD object.
<i>cvId</i>	<i>dbCellViewId</i> , the top-level cellview containing the ROD object.

Return Values

<i>rodObjectId</i>	The ROD object <i>id</i> specified by the hierarchical name <i>hierName</i> .

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodGetObjDbId

```
dbId  
rodGetObjDbId(  
    rodObjectId    objId  
);
```

Description

Returns the `dbId` of the specified ROD object. It first checks to make sure the `objId` passed represents a ROD object.

Arguments

<code>objId</code>	ROD object <code>id</code> for the object.

Return Values

<code>dbId</code>	The database <code>id</code> for the associated object.

rodGetObjName

```
String  
rodGetObjName (  
    rodObjectId      objId  
);
```

Description

Returns the hierarchical name of the specified ROD object. The name is constructed in a static string. It first checks to make sure the *objId* passed in.

Arguments

<i>objId</i>	ROD object id for the object.

Return Values

String	The hierarchical name of the associated object.

rodGetObjTransform

```
dbTransform  
rodGetObjTransform(  
    rodObjectId    objId  
);
```

Description

Returns the transform of the specified ROD object.

Arguments

<i>objId</i>	ROD object id for the object.

Return Values

<i>dbTransform</i>	The transform of the associated object.

Multipart Path Functions

rodDeleteAllMPPs

Boolean

```
rodDeleteAllMPPs (
    dbCellViewId    cvId,
    Boolean          immediateUpdate
);
```

Description

Deletes all of the multipart paths (MPPs) in the specified cellview.

Arguments

<i>cvId</i>	The <code>dbCellViewId</code> , the <code>id</code> of the top-level cellview containing the ROD object.
<i>immediateUpdate</i>	The <i>immediateUpdate</i> flag.

Return Values

Boolean	Returns TRUE if successful. If the <i>immediateUpdate</i> flag is on, it also immediately calls <code>rodiUpdate</code> . Returns FALSE otherwise.

rodExplodeMPP

Boolean

```
rodExplodeMPP(  
    dbPathId    pathId  
);
```

Description

Changes the MPP into an ordinary path and its subparts into ordinary paths and rectangles; all ROD attributes such as names and alignment are lost.

Arguments

<i>pathId</i>	The <code>dbPathId</code> of the MPP master path.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE otherwise.

rodGetMPPSubShapeMaster

dbIdm

```
rodGetMPPSubShapeMaster(  
    dbId      id  
);
```

Description

Returns the database *id* of the master path for the specified *id* if the *id* is a *dbId* for a subshape that is part of a ROD multipart path. The function returns *dbcNullId* otherwise. *dbcNullId* is returned if the specified *id* is the *dbId* for the master path of a ROD multipart path. The master path is not considered a subshape.

Arguments

<i>id</i>	The <i>dbId</i> for a subshape that is part of a ROD multipart path.

Return Values

<i>dbIdm</i>	The database <i>id</i> for the master path of a MMP for which the specified <i>id</i> identifies a subshape.

Multipart Path Subshape Generation Functions

rodStartGenMPPDbIds

```
rodMPPGenStateId  
rodStartGenMPPDbIds (  
    dbPathId      pathId  
);
```

Description

Generates the MPP subshape database *ids*.

Arguments

<i>pathId</i>	The database <i>id</i> of the MPP master path.

Return Values

rodMPPGenStateId	The MPP subshape database <i>ids</i> .

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodGenMPPDbId

```
dbId  
rodGenMPPDbId(  
    rodMPPGenStateId  stateId  
);
```

Description

Returns the a pointer to the next MPP subshape `dbId`. When done, returns `dbcNullId`.

Arguments

<i>stateId</i>	The <code>stateId</code> generated by the <code>rodStartGenMPPDbIds</code> function.

Return Values

<code>dbId</code>	The <code>dbcNullId</code> .

rodStopGen

`void`

`rodStopGen (stateId)`

Description

Frees the storage used by the generation control block.

Arguments

<i>stateId</i>	The <code>stateId</code> of the path.

Multipart Rectangle Functions

rodGetMPRSubShapeMaster

```
dbId  
rodGetMPRSubShapeMaster(  
    dbId      id  
) ;
```

Description

Returns the database *id* of the master rectangle for the specified *id* if the *id* is a *dbId* for a subpart that is part of a ROD multipart rectangle. The function returns *dbcNullId* otherwise. *dbcNullId* is returned if the specified *id* is the *dbId* for the master rectangle of a ROD multipart rectangle. The master rectangle is not considered a subpart.

Arguments

<i>id</i>	The <i>dbId</i> for a subshape that is part of a ROD multipart rectangle.

Return Values

<i>dbId</i>	The database <i>id</i> for the master rectangle of a MPR for which the specified <i>id</i> identifies a subpart.

rodExplodeMPR

Boolean

```
rodExplodeMPR(  
    dbRectId      rectId  
);
```

Description

Changes the MPR into an ordinary rectangle and its subparts into ordinary parts and rectangles; all ROD attributes such as names and alignment are lost.

Arguments

<i>rectId</i>	The <code>dbRectId</code> of the MPR master rectangle.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodReCreateMPR

Boolean

```
rodReCreateMPR(  
    dbRectId      rectId  
);
```

Description

Re-creates a MPR. Removes all old `dbIds` and creates new ones.

Arguments

<i>rectId</i>	The <code>dbRectId</code> of the MPR master rectangle.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE otherwise.

Multipart Rectangle Subshape Generation Functions

rodGetMPRRecId

```
rodMPRRecId  
rodGetMPRRecId(  
    dbId          id  
);
```

Description

Returns the MPR `recId` if the `dbId` is an MPR master.

Arguments

<i>id</i>	The <code>dbId</code> for a subpart that is part of a ROD multipart rectangle.

Return Values

<code>rodMPRRecId</code>	The MPR record <code>id</code> .

rodStartGenMPRSubParts

```
rodStartMPRSubParts(  
    dbId      Id  
);
```

Description

Allocates a `rodMPRGenState` then fills it with an array of `rodMPRSubPart` pointers

Arguments

<i>Id</i>	The database <code>id</code> of the MPR master rectangle.

Return Values

Returns a null pointer.

rodStartGenModMPRSubParts

```
rodMPRGenStateId  
rodStartGenModMPRSubParts (  
    rodMPRRecId      MPRRecId  
    dbBBox           *bBox  
);
```

Description

Modifies the MPR structures according to the new bounding box.

Arguments

<i>MPRRecId</i>	The database id of the MPR master rectangle.
<i>bBox</i>	The database id of the MPR master rectangle bounding box.

Return Values

<i>rodMPRGenStateId</i>	The MPR genState Ids.

rodGenMPRSubParts

```
*rodMPRSubPart  
rodGenMPRSubParts (  
    rodMPRGenStateId    stateId  
);
```

Description

Returns the pointer to the next MPRSubPart `dbId`. When done, returns `dbcNullId`.

Arguments

<i>stateId</i>	The <code>stateId</code> generated by the <code>rodStartGenMPRSubParts</code> function.

Return Values

<i>*rodMPRSubPart</i>	The pointer to <code>rodMPRSubPart</code> .

rodGenMPRDbId

```
dbId  
rodGenMPRDbId (  
    rodMPRGenStateId      stateId  
);
```

Description

Returns the pointer to the next MPR `dbId`.

Arguments

<i>stateId</i>	The <code>dbId</code> for a subshape that is part of a ROD multipart rectangle.

Return Values

<code>dbId</code>	The database <code>id</code> .

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodStartGenMPRDbIds

```
rodMPRGenStateId  
rodStartGenMPRDbIds (  
    dbRectId      rectId  
);
```

Description

Starts to generates the MPR subshape database `ids`.

Arguments

<i>rectId</i>	The database <code>id</code> of the MPR master rectangle.

Return Values

<code>rodMPRGenStateId</code>	The MPR <code>genState id</code> .

rodStopGenMPR

```
void  
rodStopGenMPR (  
    rodMPRenStateId      stateId  
);
```

Description

Frees the storage used by the MPR generation control block.

Arguments

<i>stateId</i>	The <code>stateId</code> of the MPR.

rodIsMPRSubShape

Boolean

```
rodIsMPRSubShape (
    dbId      id
);
```

Description

Determines if the specified *id* is a `dbId` for a subshape that is part of a ROD multipart rectangle. The master path is not considered a subshape.

Arguments

<i>id</i>	The <code>dbId</code> for the master path of a ROD multipart rectangle.

Return Values

Boolean	Returns TRUE if the specified <i>id</i> is a <code>dbId</code> for a subshape that is part of a ROD multipart rectangle. Returns FALSE otherwise. The master path is not considered a subshape; if the specified <i>id</i> is the <code>dbId</code> for the master path of a ROD multipart rectangle, FALSE is returned.

Name, Unalign, Unname Functions

rodNameShape

```
rodObjectId  
rodNameShape (  
    String      name,  
    dbShapeId   shapeId  
);
```

Description

Names the specified `shapeId` making it a ROD object.

Assigns a name to an unnamed database shape or renames a ROD shape. Creates (or updates) a ROD object containing information associated with the shape, including its name and database ID. The associated ROD object is identified by a ROD object ID. You can assign a name to any of the following shapes:

- rectangle
- polygon
- path
- line
- dot
- label
- text-display object

Note: Instances and mosaics are not shapes. You cannot rename an instance or a mosaic with this function.

You can refer to a named shape through hierarchy using its hierarchical name.

To assign a name to an unnamed shape, you must identify the shape by its database ID. You can specify a name that is unique within the cellview or let the system generate a name. If you attempt to assign a name to an object that already has a name, the system displays an error message.

Before assigning a name, you might want to verify that the string is a valid name and that it has not already been used in the cellview. To do this, use the `rodIsFigNameUnused` SKILL function.

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

Arguments

<i>name</i>	Character string or symbol specifying the name you want to assign to the shape. The name cannot contain hierarchy, where hierarchy is indicated by one or more slashes. For example, at the lowest level in the hierarchy (zero-level), you can assign the name <code>polyRect</code>. You cannot assign a name with implied hierarchy, such as <code>I1/I2/polyRect</code>. When you do not specify a name, the system determines whether the shape is a rectangle, polygon, path, line, dot, label, and/or text-display object and assigns a name indicating the shape. For rectangles, the system assigns <code>rect0</code> to the first rectangle you name in the cellview, <code>rect1</code> to the second, and so on. For polygons, the system assigns <code>polygon0</code>, <code>polygon1</code>, and so on; for paths, <code>pathn</code>. For a dot, line, label, and/or text-display object, the system assigns <code>dotn</code>, <code>linen</code>, <code>labeln</code>, or <code>textn</code>, respectively. For a shape other than these, the system issues a warning message and the function fails. Default: <code>rectn</code>, <code>polygonn</code>, <code>pathn</code>, <code>dotn</code>, <code>linen</code>, <code>labeln</code>, or <code>textn</code>, as determined by the system
<i>shapeId</i>	Database of unnamed shape. (Instances and mosaics are not shapes.) This argument is required.

Return Values

<code>rodObjectId</code>	The ROD object <code>id</code> for the named shape.

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodUnAlign

Boolean

```
rodUnAlign(  
    rodObjectId    objId1,  
    rodObjectId    objId2  
);
```

Description

When you specify only one ROD object *id*, removes all alignments for the object specified. When you specify two ROD object *ids*, removes only the alignment between the two objects.

Arguments

<i>objId1</i>	ROD object <i>id</i> for the first or only object for which you want to remove alignments. When <i>objId2</i> is specified, only the alignments between the two objects are removed. When <i>objId2</i> is not specified, all alignments for <i>objId1</i> are removed. This argument is required.
<i>objId2</i>	ROD object ID for the second object for which you want to remove alignments. When specified, only the alignment between <i>objId2</i> and <i>objId1</i> are removed.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE otherwise.

rodUnNameShape

Boolean

```
rodUnNameShape (
    rodObjectId      objId
);
```

Description

Removes the ROD name from the `dbId` of the specified ROD object, so afterwards it is no longer a named shape. The ROD object `id` for the shape and all other ROD information is deleted. If the object is an instance, the instance keeps its name but all other ROD information is deleted. Also removes all ROD alignment constraints that reference the shape.

Arguments

<code>objId</code>	ROD object <code>id</code> that identifies the shape for which you want to remove the name.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE otherwise.

Handle Functions

rodGetHandle

Boolean

```
rodGetHandle (
    rodObjectId    objId,
    String          name,
    rodHandleType   *pHandleType,
    rodHandleValue  *pHandleValue
);
```

Description

Implements the ROD user function `rodGetHandle`.

The ROD user function `rodGetHandle` returns the value of a system- or user-defined handle. You specify the handle name and the ROD object `id` for the object with which the handle is associated.

For Boolean handles, the value can be either `t` or `nil`. When the value is `nil`, `rodGetHandle` returns an ambiguous result: `nil` could mean that the handle was not found or that the value of the handle is `nil`. To avoid ambiguity for Boolean handles, verify that the handle exists with the `rodIsHandle` function before you try to access it.

Arguments

<i>objId</i>	ROD object <code>id</code> for the object with which the handle is associated.
<i>name</i>	Character string or symbol specifying the name of the handle whose value is returned.
<i>pHandleType</i>	The type of handle returned.

Note: System-Defined Handles for ROD Objects

Type of Handle	Applies to...
Bounding Box Handles	All ROD objects except dots, labels, and text display objects

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

Type of Handle	Applies to...
Bounding Box Width and Length Handles	All ROD objects except dots, labels, and text display objects
ppBBox	ROD multipart paths
Segment Point Handles	ROD rectangles, polygons, and paths
Segment Length Handles	ROD rectangles, polygons, and paths

Note: User-defined handles for ROD objects

You can define your own handles to store points, calculations, and other information. When you define a new handle, you specify the name (or let it default) and assign a value. The values of user-defined handles are stored in the database. The information stored can have any of the following data types:

point	integer
Boolean	floating-point number
string	SKILL expression

If you let the name of your new handle default, the system assigns a name unique within the cellview, as follows: `handle0`, `handle1`, `handle2`, and so on.

pHandleValue The value of the handle returned.

Return Values

Boolean Returns TRUE if successful. Returns FALSE otherwise.

rodGenStatId

```
rodGenStateId  
rodStartGenNamedShape(  
    dbCellViewId    cvId  
);
```

Description

Allocates a `rodGenState` control block and fills it in, setting things up for `rodGenNamedShape`, which returns one `rodObjectId` at a time when it is called.

Arguments

<i>cvId</i>	The cellview <code>id</code> in which to generate the ROD named shapes.

Return Values

<code>rodGenStateId</code>	Unique generator state <code>id</code> for use with subsequent calls to <code>rodGenNamedShape</code> .

Cadence Integrators Toolkit Database Reference

CDBA Relative Object Design Functions

rodGenNamedShape

Boolean

```
rodGenNamedShape (
    rodGenStateId    stateId,
    rodObjectId      objId
);
```

Description

Returns TRUE if there was another `rodObjectId` generated to fill in the return argument `objId`. Returns FALSE otherwise.

Arguments

<i>stateId</i>	The <code>stateId</code> generated by the <code>rodGenNamedShape</code> function.
<i>objId</i>	The ROD object id.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE otherwise.

CDBA User Database Extensions

The database can be extended using properties and groups. You can notify an application of changes in the state of the database.

Find more information about database extensions in the following sections:

■ Properties on page 1093

- ❑ Property Types on page 1093
- ❑ Property Creation and Modification Functions on page 1095
- ❑ Property Information Replacement Functions on page 1099
- ❑ Property Information Retrieval Functions on page 1102
- ❑ Predicate Functions for Properties on page 1106
- ❑ Property Deletion Functions on page 1107
- ❑ Property Generator Functions on page 1108

■ Groups on page 1110

- ❑ Group Attributes on page 1110
- ❑ Groups as Sets or Collections on page 1110
- ❑ Deleting Groups on page 1111
- ❑ Group Member Order on page 1111
- ❑ Group Names on page 1111
- ❑ Group Members on page 1112
- ❑ Group Creation Functions on page 1112
- ❑ Group Modification Functions on page 1113
- ❑ Group Information Retrieval Functions on page 1116

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

- ❑ [Predicate Functions for Groups](#) on page 1118
- ❑ [Group Deletion Functions](#) on page 1118
- ❑ [Group ID Generator Functions](#) on page 1120
- [Pin Group Guides](#) on page 1124
 - ❑ [Functions for Adding, Removing, and Accessing Pins](#) on page 1124
 - ❑ [Functions for Accessing Pin Group Guide Attributes](#) on page 1126
 - ❑ [Functions for Specifying Pin Group Guide Constraints](#) on page 1128
 - ❑ [Area Pin Group Guides Functions](#) on page 1133
 - ❑ [Edge Pin Group Guide Functions](#) on page 1136
 - ❑ [Pin Group Guide Functions for Other Objects](#) on page 1138

Properties

All database objects can have any number of properties attached to them. Each property is represented by a `dbPropId` and is defined by a name and a value. When a database object is deleted, all properties attached to this object are also automatically deleted.

No two properties attached to the same object can have the same name. Also, properties of the same name on different objects exist independently of each other (that is, there is no relationship between property X on object A and property X on object B). Therefore, the values, ranges, or enumerations can be changed on one property without affecting other properties of the same name.

Property Types

The three major types of properties are scalar, range, and enumerated.

Scalar Properties

A scalar property can have one of the following types of values. Each of these types is represented by a predefined enumerated constant of the type `dbPropType`.

- A Boolean-valued property can have one of two values: TRUE or FALSE.
- A float-valued property is represented by a 32-bit floating-point number.
- An integer-valued property is represented by a 32-bit integer number.
- A string-valued property is a pointer to a null-terminated array of characters.
- An `il` expression-valued property is not available in ITKDB.
- A file name-valued property is the path of a file and is represented by a string.
- An NLP-expression-valued property is represented by a string containing an NLP (netlist property) expression. These expressions are interpreted and used by Cadence® netlist generators.
- A time-valued property is represented by an unsigned 32-bit integer containing the number of seconds elapsed from 00:00:00 GMT, Jan. 1, 1970.
- A hierarchical property has other properties attached to it but has no value in itself.
- An IL list-valued property is not available in ITKDB.

A property value is represented by a predefined union of type `dbPropValue`:

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

```
union dbPropValue {
    Boolean    aBoolean;
    float      aFloat;
    int        aInt;
    String     aString;
    String     aFileName;
    String     aILExpr; /* not available in ITK:Database */
    String     aNLPEmr;
    long       aTime;
    char       *aILLlist; /* not available in ITK:Database */
};
typedef union dbPropValue dbPropValue;
```

The following table lists the scalar property types and their enumerated constants.

Scalar Property Types

Type	Enumerated constants in type dbPropType
Boolean	dbcBooleanType
file name	dbcFileNameType
float	dbcFloatType
integer	dbcIntType
String	dbcStringType
IL expression	dbcILExprType (not available in ITKDB)
NLP expression	dbcNLPEmrType
time	dbcTimeType
hierarchical property	dbcHierPropType
IL list	dbcILLlistType (not available in ITKDB)

Range Properties

A range property must be an integer, a floating-point number, or a time value. It also contains upper and lower boundary values that define the inclusive range of the property value. The upper and lower boundaries, as well as the range value, are defined by a `dbPropValue` union.

Enumerated Properties

An enumerated property contains a string value and an enumerated set (list) of strings. The value assigned to this property must be a string in this enumerated set.

Property Creation and Modification Functions

The following functions can be used to create or modify properties.

In all examples specified in this section,

- `id` (or `ID` in text) is defined as a generic `dbId`.
- `value` is defined as a `dbPropValue` union.
- `propId` is defined as a `dbPropId`.

dbCreateProp

```
dbPropId
dbCreateProp(
    dbId id, String propName, dbPropType propType, dbPropValue propValue
);
```

Creates a scalar property belonging to the database object specified by the `ID` argument. The scalar property is given the name `propName`, the type defined in `propType`, and the value defined in `propValue` (where `propType` and `propValue` are defined). The function returns the `dbPropId` if the property was created successfully. The function returns `NULL` if a property named `propName` already exists on this object or if an error occurred.

Example

```
value.aInt = 5;
propId = dbCreateProp(id, "width", dbcIntType, value);
value.aTime = time(0);
propId = dbCreateProp(id, "timeCreated", dbcTimeType, value);
value.aILExpr = "if((x > 5) 1 0)";
propId = dbCreateProp(id, "ilFunc", dbcILExprType, value);
```

The property value must be passed as a `dbPropValue`. A scalar value passed directly can cause problems on some systems. In the following line of code, the value of 1 for the `dbcIntType` that is passed is not portable to any system.

```
/* The following line is NON-PORTABLE! */
propId = dbCreateProp(id, "ilFunc", dbcIntType, 1);
```

Rather, the line should read

```
propId = dbCreateProp(id, "ilFunc", dbcIntType, value);
```

dbCreateHierProp

```
dbPropId
```

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

```
dbCreateHierProp(  
    dbId id, String propName  
);
```

Creates a hierarchical property belonging to the database object specified by the ID argument. The hierarchical property is named `propName`. The function returns the `dbPropId` if the property was created successfully. The function returns NULL if a property `propName` already exists on this object or if an error occurred.

Example

```
propId = dbCreateProp(id, "parameters");
```

dbCreateRangeProp

```
dbPropId  
dbCreateRangeProp(  
    dbId id, String propName, dbPropType propType, dbPropValue propValue,  
    dbPropValue lowerBound, dbPropValue upperBound  
);
```

Creates a range property belonging to the database object specified by the ID argument. The range property is given the name `propName`, the type defined in `propType`, and the value defined in `propValue`.

The three types of range properties are integer, float, or time (`dbcIntType`, `dbcFloatType`, or `dbcTimeType`, respectively). This function specifies the `lowerBound` and `upperBound` of the range and defines them as `dbPropValue` unions. The boundaries must be the same type as the property value.

This function returns the `dbPropId` if the property was created successfully. The function returns NULL if a property named `propName` already exists on this object, if the specified value is not within the specified boundaries, or if an error occurred.

Example

```
value.aFloat = 32.5;  
lowerBound.aFloat = 0.0;  
upperBound.aFloat = 50.0;  
propId = dbCreateRangeProp(id, "length", dbcFloatType, value, lowerBound,  
    upperBound);
```

Example

```
value.aInt = 33;
```


Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

```
lowerBound.aInt = 0;
upperBound.aInt = 100;
propId = dbCreateRangeProp(id, "numberOfRects", dbcIntType, value, lowerBound,
upperBound);
```

dbCreateEnumProp

dbPropId

```
dbCreateEnumProp(
    dbId id, String propName, String propString, unsigned int nStrings,
    String arrayOfStrings
);
```

Creates an enumerated property belonging to the database object specified by the ID argument. The enumerated property is named `propName` and given the value of `propString`. The number of strings in the enumerated set is specified in `nStrings`. The enumerated set is defined by the strings stored in `arrayOfStrings`. The function returns the `dbPropId` if the property was created successfully. The function returns NULL if a property named `propName` already exists on this object, if the specified value is not in the specified enumerated set, or if another error occurred.

Example

```
/*
 * Create an enumerated property with current value
 * of "Wednesday," where valid values are defined to
 * be days of the week.
 */

static char *day = {
    "Sunday",
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday"
};
propId = dbCreateEnumProp(id, "dayInWeek", "Wednesday",
    7, day);
```

dbCopyProp

Boolean

```
dbCopyProp(
    dbId srcId, dbId destId
);
```

Copies all properties attached to the database object identified by `srcId` to the database object identified by `destId`. The copy is performed recursively for all hierarchical properties found.

dbCopySingleProp

```
dbPropId  
dbCopySingleProp(  
    dbId srcPropId, dbId destId  
);
```

Copies a specified property identified by the `srcPropId` to the destination object identified by `destId`. The function returns the destination `propId`. This function is different from `dbCopyProp`, which copies all properties from the source object to the destination object. The copy is performed recursively for a hierarchical property.

dbMergeProp

```
Boolean  
dbMergeProp(  
    dbId srcId, dbId destId  
);
```

Merges all properties attached to the database object identified by `srcId` to the database object identified by `destId`. That is, the function copies the source property only if the property does not exist on the destination object. If the property exists, the property is not overwritten. This is different from `dbCopyProp`, which overwrites the destination property if one exists.

dbMergeSingleProp

```
dbPropId  
dbMergeSingleProp(  
    dbId srcPropId, dbId destId  
);
```

Merges the property specified by `srcPropId` to the database object identified by `destId`. The difference between this function and `dbCopySingleProp` is that if the property already exists in the destination object, this function does not overwrite it. The function just returns the `propId`. However, `dbCopySingleProp` overwrites the property in the destination object.

dbResetPropList

Boolean

```
dbResetPropList(  
    dbId          id,  
    unsigned int  nProps,  
    dbParamRec    *propList  
);
```

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>id</i>	Database ID of the object whose properties are being replaced.
<i>nprops</i>	Number of properties in the <i>propList</i> .
<i>propList</i>	List of properties.

Return Value

<i>Boolean</i>	TRUE, if the list of properties on a given object are reset with the specified list of properties. FALSE, in case of failure.

Property Information Replacement Functions

The following functions retrieve information about properties and potentially overwrite existing information.

dbReplaceProp

```
dbPropId  
dbReplaceProp(  
    dbId id, String propName, dbPropType propType, dbPropValue propValue  
);
```

Creates a scalar property belonging to the database object specified by the ID argument (like `dbCreateProp`), except that if a property named `propName` already exists on the object, `propType` and `propValue` overwrite its type and value. The function returns the `dbPropId` of the property created or replaced if the action was successful. Otherwise, the function returns `NULL`.

Note: `dbReplaceProp` always overwrites a hierarchical, range, or enumerated property with a scalar property and does not issue a warning.

dbReplacePropList

```
Boolean  
dbReplacePropList(  
    dbId id, unsigned int nProps, dbParamRec *propList  
);
```

Replaces a list of properties of a database object specified by `id` with the list specified by `propList`. The number of properties is specified by `nProps`. If a property already exists, update its type and value, otherwise, create it.

Note: This function should not be used to replace properties of a pCell instance because the pCell parameters will not be evaluated by this function. To update pCell parameters, use `dbReplaceInstParamList`.

dbReplaceInstParamList

```
Boolean  
dbReplaceInstParamList(  
    dbId id, unsigned int nParams, dbParamExtRec *params  
);
```

Replaces a list of parameters for the instance with the list specified by `params`. The number of parameters is specified by `nParams`. The parameters should contain all parameters including a parameter having a default value. If a parameter already exists, update the type and value, otherwise, create it.

Note: Use this function to update pCell parameters instead of the `dbReplacePropList` function.

dbReplaceHierProp

```
dbPropId  
dbReplaceHierProp(  
    dbId id, String propName  
);
```

Creates a hierarchical property belonging to the database object specified by the ID argument (like `dbCreateHierProp`), except that if a property named `propName` already exists on the object, `dbCPropertyType` overwrites its type and the value is deleted. The function returns the `dbPropId` of the property created or replaced if the action was successful. Otherwise, the function returns `NULL`.

Note: `dbReplaceHierProp` always overwrites a scalar, range, or enumerated property with a hierarchical property and does not issue a warning.

dbReplaceRangeProp

```
dbPropId  
dbReplaceRangeProp(  
    dbId id, String propName, dbPropType propType, dbPropValue propValue,  
    dbPropValue lowerBound, dbPropValue upperBound  
);
```

Creates a range property belonging to the database object specified by the ID argument (such as `dbCreateRangeProp`), except that if a property named `propName` already exists on the object, the property's `propType`, `propValue`, `lowerBound`, and `upperBound` are overwritten by the given `propType`, `propValue`, `lowerBound`, and `upperBound`.

The function returns the `dbPropId` of the property created or replaced if the action was successful. Otherwise, it returns `NULL`.

Note: `dbReplaceRangeProp` always overwrites a scalar, hierarchical, or enumerated property with a range property and does not issue a warning.

dbReplaceEnumProp

```
dbPropId  
dbReplaceEnumProp(  
    dbId id, String propName, String propString, unsigned int nStrings,  
    String arrayOfStrings  
);
```

Creates an enumerated property belonging to the database object specified by the ID argument (like `dbCreateEnumProp`), except that if a property named `propName` already

exists on the object, `propString`, `nStrings`, and `arrayOfStrings` overwrite its type and value. The function returns the `dbPropId` of the property created or replaced if the action was successful. Otherwise, the function returns `NULL`.

Note: `dbReplaceEnumProp` always overwrites a scalar, hierarchical, or range property with an enumerated property and does not issue a warning.

Property Information Retrieval Functions

The following functions retrieve information about properties.

dbGetProp

Boolean

```
dbGetProp(  
    dbPropId propId, String pPropName, dbPropType pPropType, dbPropValue  
    pPropValue  
);
```

Finds the property specified by `propId` and returns the name, type, and value of this property in the specified `PropName`, `PropType`, and `PropValue` arguments when the value type equals a `String` type. If you need a permanent copy of the property name or the property value, you must explicitly copy the value into your own storage area. The function returns `TRUE` if the retrieval was successful. The function returns `FALSE` if `propId` is not valid or if another error occurred.

Note: `dbGetProp` returns any type of property, including hierarchical, range, and enumerated. In the case of a hierarchical property, `PropValue` is undefined.

Example

```
String      name;  
dbPropType  type;  
dbPropValue value;  
/* Print out the value of propId */  
if (dbGetProp(propId, &name, &type, &value)) {  
    printf("property name = %s\n", name);  
    switch (type) {  
        case dbcBooleanType:  
            printf("boolean value is %s\n",  
                (value.aBoolean) ? "TRUE" : "FALSE");  
            break;  
        case dbcIntType:  
            printf("integer value is %d\n", value.aInt);  
            break;  
        case dbcFloatType:  
            printf("floating point value is %f\n",  
                value.aFloat);  
    }
```

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

```
        break;
    case dbcStringType:
        printf("string value is %s\n", value.aString);
        break;
    case dbcFileNameType:
        printf("file name value is %s\n", value.aFileName);
        break;
    case dbcILExprType:
        printf("IL expression value is %s\n", value.aILExpr);
        break;

case dbcNLPEExprType:
    printf("NLP expression value is %s\n", value.
        aNLPEExpr);
    break;
case dbcTimeType:
    printf("time value is %s\n", ctime(&value.aTime));
    break;
case dbcHierPropType:
    printf("hierarchical property value is
        undefined\n");
    break;
case dbcILListType:
    {
        char printBuffer[MAX_LIST_STRING_SIZE];
        ilSprintf(printBuffer, MAX_LIST_STRING_SIZE,
            ilCons(ilMakeString("%L"), value.aILList));
        printf("IL list value is %s\n", printBuffer);
    }
    break;
default:
    printf("ERROR: unknown property value type %d\n",
        type);
    break;
}
}
```

dbGetPropByName

dbPropId

```
dbGetPropByName (
    dbId id, String propName, dbPropType pPropType, dbPropValue pPropValue
);
```

Searches for the property named `propName` belonging to the database object specified by `id` and returns the type and value of this property in the specified `PropType` and `PropValue` arguments. The function returns the `dbPropId` if the retrieval was successful. The function returns NULL if the property does not exist on this object or if another error occurred.

`dbGetPropByName` returns any type of property, including hierarchical, range, and enumerated. In the case of a hierarchical property, `PropValue` is undefined.

dbGetPropRange

Boolean

```
dbGetPropRange(  
    dbPropId propId, String pPropName, dbPropType pPropType, dbPropValue  
    pPropValue, dbPropValue pLowerBound, dbPropValue pUpperBound  
);
```

Finds the range property specified by `propId` and returns the name, type, value, and range boundaries of this property in the specified `PropName`, `PropType`, `PropValue`, `LowerBound`, and `UpperBound` arguments. If you need a permanent copy of the property name, you must explicitly copy the value into your own storage area. The function returns TRUE if the retrieval was successful. The function returns FALSE if `propId` is not valid, if the property is not a range property, or if another error occurred.

Example

```
String      name;  
dbPropType  type;  
dbPropValue value, min, max;  
/* Print out the value and range of propId */  
if (dbIsRangeProp(propId) &&  
    dbGetPropRange(propId, &name, &type, &value, &min,  
                   &max)) {  
    switch (type) {  
    case dbcFloatType:  
        printf("prop %s = %f; range is from %f to %f\n",  
               name, value.aFloat, min.aFloat, max.aFloat);  
        break;  
    case dbcIntType:  
        printf("prop %s = %d; range is from %d to %d\n",  
               name, value.aInt, min.aInt, max.aInt);  
        break;  
    case dbcTimeType:  
        printf("prop %s = %s; range is from %s to %s\n",  
               name, ctime(&value.aTime),  
               ctime(&min.aTime), ctime(&max.aTime));  
        break;  
    default:  
        printf("range prop %s has invalid type\n", name);  
        break;  
    }  
}
```

dbGetPropEnum

Boolean

```
dbGetPropEnum(  
    dbPropId propId, String pPropName, String pPropString, unsigned int  
    pNStrings, String pArrayOfStrings  
);
```


Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

Finds the enumerated property specified by `propId` and returns the name and value of this property in the specified `PropName` and `PropString` arguments. The argument `NStrings` returns the number of strings in the enumerated set. The argument `ArrayOfStrings` returns the enumerated set (an array of strings).

If you need a permanent copy of the property name, value, or the enumerated set, you must explicitly copy the value into your own storage area. It returns `TRUE` if the retrieval was successful. It returns `FALSE` if `propId` is not valid, if the property is not an enumerated property, or if another error occurred.

Example

```
...
String      name, value, *enumSet;
unsigned int num;
/* Print the value and enumerated set of propId */

if (dbIsEnumProp(propId) && dbGetPropEnum(propId, &name,
                                         &value, &num, &enumSet)) {
    printf("prop %s = %s, enumerated set is:\n", name, value);
    for (i = 0; i < num; i++) {
        printf("\t%s\n", *enumSet++);
    }
}
```

Predicate Functions for Properties

The following functions return predicate (Boolean) information about properties.

dbObjHasProp

Boolean

```
dbObjHasProp(  
    dbId id  
);
```

Returns TRUE if the object specified has one or more properties associated with it. Otherwise, the function returns FALSE.

dbIsScalarProp

Boolean

```
dbIsScalarProp(  
    dbPropId propId  
);
```

Returns TRUE if `propId` refers to a scalar property. Otherwise, the function returns FALSE.

dbIsHierProp

Boolean

```
dbIsHierProp(  
    dbPropId propId  
);
```

Returns TRUE if `propId` refers to a hierarchical property. Otherwise, the function returns FALSE.

dbIsRangeProp

Boolean

```
dbIsRangeProp(  
    dbPropId propId  
);
```

Returns TRUE if `propId` refers to a range property. Otherwise, the function returns FALSE.

dbIsEnumProp

Boolean

```
dbIsEnumProp(  
    dbPropId propId  
);
```

Returns TRUE if `propId` refers to an enumerated property. Otherwise, the function returns FALSE.

Property Deletion Functions

The following functions delete properties attached to database objects.

dbDeleteProp

Boolean

```
dbDeleteProp(  
    dbId id, dbPropId propId  
);
```

Deletes the property specified by `propId`, which is attached to the object specified by ID. The function returns TRUE if the property exists on the object and is successfully deleted. The function returns FALSE if either the property does not belong to the object or another error occurs.

dbDeletePropByName

Boolean

```
dbDeletePropByName(  
    dbId id, String propName  
);
```

Deletes the property named `propName` belonging to the database object specified by the ID argument. The function returns TRUE if a property of that name exists on the object and is successfully deleted. The function also returns TRUE if the object does not have a property of that name. Thus, you can use `dbDeletePropByName` to delete properties of a specified name without having to first test if all objects you are going to operate on have properties of that name attached to them. The function returns FALSE if an error occurs.

Property Generator Functions

The following pair of functions work together to generate properties.

dbStartGenProp

```
dbGenStateId
dbStartGenProp(
    dbId id
);
```

The `dbStartGenProp` function works with the `dbGenProp` function to generate all properties associated with the database object specified by `id`. Because the function `dbGenProp` is not recursive, recursion on hierarchical properties must be handled explicitly as demonstrated in the `dbGenProp` example.

dbGenProp

```
Boolean
dbGenProp(
    dbGenState state, dbPropId pPropId
);
```

Generates all properties associated with the database object specified by `id` in the `dbStartGenProp` function. Because `dbGenProp` is not recursive, recursion on hierarchical properties must be handled explicitly as demonstrated in the following example.

Example

```
processAllProps(id)
    dbId id;
{
    dbGenStateId      state; /* generator state */
    String            propName;
    dbPropId          propId;
    dbPropValue       propValue;
    dbPropType        propType;

    state = dbStartGenProp(id);
    while (dbGenProp(state, &propId)) {
        dbGetProp(propId, &propName, &propType, &propValue);
        if (propType == dbcHierPropType) {
            /* Process hierarchical property name */
            ...
            processAllProps((dbId)propId); /* recurse */
        }
        else {
            /* Process non-hierarchical property name and
             * value */
            ...
        }
    }
}
```

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

```
    }  
  }  
  dbStopGen(state);  
}
```

Groups

A group is a mechanism by which various objects in the same cellview can be related to one another. The user determines the purpose of the relationship. Groups provide a way to create and maintain the relationships in the database. Just as properties provide a way for the user to create user-defined attributes, groups provide a way for the user to create user-defined relationships.

A group is represented by a `dbGroupId` and has a name, group members, and attributes that control how the database maintains the group. The `dbCreateGroup` function creates groups. A `dbGroupMemId` represents every occurrence of an object as a member of a group.

Group Attributes

The attributes of a group include

- The group type (either a set or a collection)
- Whether the members of the group are ordered
- Whether the group name is unique within the cellview that contains it
- A deletion control attribute

Groups as Sets or Collections

A group can be either a set or a collection. A set is a group in which an ID can appear only once. A collection is a group in which an ID can appear more than once. The type of group is represented by a predefined enumerated constant in the enumerated type `dbGroupType`. An ID can be contained in more than one group.

The following table gives the enumerated constants for the set and collection group types.

Group Types

Name	Enumerated constants in type <code>dbGroupType</code>
Set	<code>dbcGroupSet</code>
Collection	<code>dbcGroupCollection</code>

Deleting Groups

The `deleteWhen` attribute indicates the conditions under which a group is deleted. This attribute is represented by a predefined enumerated constant of the enumerated type `dbDeleteWhenType`.

A group can have one of three `deleteWhen` attribute values:

- `deleteNever` indicates the group must never be deleted automatically, even when it contains no members.
- `deleteFirst` indicates the group is automatically deleted when any member of the group is deleted.
- `deleteLast` indicates the group is automatically deleted when it contains no members.

Groups created with `dbCreateGroup` have the `deleteWhen` attribute set to `deleteNever` by default. Use `dbSetGroupDeleteWhen` to set a different value after the group is created.

The following table gives the enumerated constants for the three `deleteWhen` attribute values.

Group deleteWhen Types

Name	Enumerated constants in type <code>dbGroupDeleteWhenType</code>
<code>deleteNever</code>	<code>dbcDeleteNever</code>
<code>deleteFirst</code>	<code>dbcDeleteFirst</code>
<code>deleteLast</code>	<code>dbcDeleteLast</code>

Group Member Order

A group can be ordered or unordered. An ordered group implies there is significance to the order of the group members stored in the group. You establish and maintain the order using calls to add ids to a group and calls to move group members within the ordered group. You specify the order when generating members of an ordered group.

Group Names

Each group has a name, which might or might not be unique within the cellview containing the group. If a group is designated to have a unique name, then no other group in the same cellview can share its name.

Group Members

Because the `collection` type group can contain multiple occurrences of the same object, `dbGroupMemId` uniquely identifies each occurrence of an ID in a group. When an ID is added to a group by calling the `dbAddIdToGroup` function, the function returns a `dbGroupMemId` type value.

An object can be a member of more than one group. An object can even be a member of two groups with the same name (if the groups were created without the unique name restriction). If a group is a collection, an object can belong to the same group more than once.

Group Creation Functions

The following function creates a group.

dbCreateGroup

```
dbGroupId
dbCreateGroup(
    dbCellViewId cellViewId, String name, dbGroupType groupType, Boolean ordered,
    Boolean uniqueName
);
```

Creates a group in the specified cellview. If you specify a group to have a unique name (if `uniqueName` is `TRUE`) and a group with the same name already exists in the specified cellview, no group is created and `dbcNullGroupId` is returned. By default, the group created is of the groupDef `unrestricted` type.

Note: The `deleteWhen` attribute is set to `dbcDeleteNever` by default. To change the value, use the function `dbSetGroupDeleteWhen`.

dbCreateGroupWithDef

```
dbGroupId
dbCreateGroupWithDef(
    dbCellViewId cellViewId, const char *groupDefName, const char *groupName,
    dbGroupType groupType, Boolean ordered, Boolean uniqueName
);
```

Creates a group of the specified group definition in the specified cellview. If the `groupDefName` is an invalid string or the specified group definition is not found, no group is created and `dbcNullGroupId` is returned. The group is created only if the specified group

definition is found, either as an OpenAccess built-in groupDef or as a user-defined groupDef that exists in that session.

Valid OpenAccess built-in groupDefs include `unrestricted`, `diffNetPair`, `net`, `matchedNet`, `nestedNet`, and `symmtericNet`.

Group Modification Functions

The following functions modify groups.

dbAddIdToGroup

```
dbGroupMemId
dbAddIdToGroup (
    dbGroupId groupId, dbId id
);
```

Includes the ID as a member of the specified group. If the group is `ordered`, it adds the ID to the end of the group. The `groupId` and the ID must belong to the same cellview. If the group is type `dbcGroupSet`, the function fails if the ID is already a member of the group. If the function succeeds, it returns a `dbGroupMemId`, which you use to uniquely identify this reference to the specified ID in the group. Otherwise, the function returns `dbcNullGroupMemId`.

Note: The following ID types cannot be added to any group:

- `dbGroupMemId` (group members)
- `dbInstHeaderId` (instance headers)

dbDeleteGroupMemFromGroup

```
Boolean
dbDeleteGroupMemFromGroup (
    dbGroupMemId groupMemId
);
```

Deletes a particular group member from the group to which it belongs. The function returns `TRUE` if `groupMemId` refers to a valid group member.

Note: If the `deleteWhen` attribute has a value of `deleteFirst` and `TRUE` is returned, the group is also deleted. If the group is empty from being deleted and the `deleteWhen` attribute has the value `deleteLast`, the group is also deleted. Otherwise, the group remains in the database.

dbDeleteIdFromGroup

Boolean

```
dbDeleteIdFromGroup(  
    dbGroupId groupId, dbId id  
);
```

Deletes all references to the specified ID from the specified group. The group ID (specified by `groupId`) and the ID (specified by `id`) must belong to the same cellview. If the ID occurs more than once in the group and you want to delete a particular occurrence of the ID from the group, you must call `dbDeleteMemFromGroup`.

Note: If the `deleteWhen` attribute has a value of `deleteFirst` and TRUE is returned, the group is also deleted. If the group is empty because its members are deleted and the `deleteWhen` attribute has the value `deleteLast`, the group is also deleted. Otherwise, the group remains in the database.

dbDeleteIdFromAllGroup

Boolean

```
dbDeleteIdFromAllGroup(  
    dbId id  
);
```

Deletes all references to the specified ID from all groups that reference the ID. This function is automatically called when an object is deleted from the database.

Note: If FALSE is returned, some references might have been deleted, as well as some groups. You can prevent this situation by setting the `deleteWhen` attribute to `deleteNever` for those groups. If TRUE is returned, all groups containing the ID and that had the `deleteWhen` attribute set to `deleteFirst` are deleted. All groups that had the `deleteWhen` attribute set to `deleteLast` (which were made empty by deleting the ID) are also deleted.

dbMoveGroupMem

Boolean

```
dbMoveGroupMem(  
    dbGroupMemId groupMemId, dbGroupMemId afterGroupMemId  
);
```

Moves `groupMemId` to just behind `afterGroupMemId` and returns TRUE if both group members belong to the same group and the group is ordered. If `groupMemId` equals `afterGroupMemId`, nothing is done and the function returns TRUE. To move `groupMemId` to the head of the group, set `afterGroupMemId` equal to `dbcNullGroupMemId`.

dbSetGroupType

Boolean

```
dbSetGroupType(  
    dbGroupId groupId, dbGroupType type  
);
```

Sets the type of the group identified by `groupId`. If the group type is changed from `dbcGroupCollection` to `dbcGroupSet` and an ID appears in the group more than once, the function fails and returns `FALSE`.

dbSetGroupOrdered

Boolean

```
dbSetGroupOrdered(  
    dbGroupId groupId, Boolean ordered  
);
```

Sets the ordered attribute for the group identified by `groupId`, indicating whether the group is ordered or unordered. If `groupId` is not valid, the function fails and returns `FALSE`.

dbSetGroupUniqueName

Boolean

```
dbSetGroupUniqueName(  
    dbGroupId groupId, Boolean uniqueName  
);
```

Sets the unique name attribute for `groupId`, indicating whether or not the group name must be unique in the cellview containing the group. If `uniqueName` is `TRUE` and there is another group with the same name in the cellview containing the group, the function fails and returns `FALSE`.

dbSetGroupDeleteWhen

Boolean

```
dbSetGroupDeleteWhen(  
    dbGroupId groupId, dbDeleteWhenType deleteWhen  
);
```

Sets the `deleteWhen` attribute for `groupId`, indicating whether the group must be automatically deleted when any member is deleted or when the group becomes empty.

Group Information Retrieval Functions

The following functions retrieve information about groups.

dbGetGroupName

```
String  
dbGetGroupName (  
    dbGroupId groupId  
);
```

Returns the name of the group as a cellview string. Returns `dbcNullString` if `groupId` is invalid.

dbGetGroupGroupDefName

```
String  
dbGetGroupGroupDefName (  
    dbGroupId groupId  
);
```

Returns the name of the group definition of the specified group as a string. Returns `dbcNullString` if `groupId` is invalid.

dbGetGroupByName

```
dbGroupId  
dbGetGroupByName (  
    dbCellViewId cellViewId, String name  
);
```

Searches for the group `name` belonging to the specified cellview. Returns the `dbGroupId` if the group exists. Returns `dbcNullGroupId` if the group does not exist in the cellview.

dbGetGroupType

```
dbGroupType  
dbGetGroupType (  
    dbGroupId groupId  
);
```

Returns the type of the group identified by `groupId`. If `groupId` is not valid, `dbcUnknownGroupType` is returned.

dbGetGroupOrdered

Boolean

```
dbGetGroupOrdered(  
    dbGroupId groupId  
);
```

Returns TRUE if the group is `ordered`. Returns FALSE if `groupId` is invalid or if the group is `unordered`.

dbGetGroupUniqueName

Boolean

```
dbGetGroupUniqueName(  
    dbGroupId groupId  
);
```

Returns TRUE if the group must have a unique name. Returns FALSE if `groupId` is invalid or if other groups in the same cellview can use the same name.

dbGetGroupDeleteWhen

dbDeleteWhenType

```
dbGetGroupDeleteWhen(  
    dbGroupId groupId  
);
```

Returns the `deleteWhen` attribute of the group unless `groupId` is invalid, in which case it returns `dbcUnknownDeleteWhenType`.

dbGetGroupMemGroup

dbGroupId

```
dbGetGroupMemGroup(  
    dbGroupMemId groupMemId  
);
```

Returns the `groupId` for the group that contains the group member identified by `groupMemId`, or returns `dbcNullGroupId` if `groupMemId` is invalid.

dbGetGroupMemId

dbId

```
dbGetGroupMemId(  
    dbGroupMemId groupMemId  
);
```

Returns the `dbId` of the database object referred to by `groupMemId` or returns `dbcNullId` if `groupMemId` is invalid.

Predicate Functions for Groups

The following function retrieves predicate information about groups; that is, it returns the result of a query about the group.

```
Boolean  
dbIsGroupEmpty(  
    dbGroupId groupId  
);
```

Returns TRUE if the group identified by `groupId` does not contain any group members. Returns FALSE if `groupId` is invalid or if there are members in the group.

Group Deletion Functions

The following functions delete groups according to the criteria specified.

dbDeleteGroup

```
Boolean  
dbDeleteGroup(  
    dbGroupId groupId  
);
```

Deletes the group referred to by `groupId` and returns TRUE if successful. If `groupId` is invalid or the delete fails for any reason, the function returns FALSE.

dbDeleteGroupByName

```
Boolean  
dbDeleteGroupByName(  
    dbCellViewId cellViewId, String name  
);
```

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

Deletes all groups in the cellview identified by `cellViewId` named `name`. Returns `TRUE` even if no groups were deleted because no groups in the cellview were named `name`. Returns `FALSE` if `cellViewId` is invalid or `name` is `dbcNullString`.

Group ID Generator Functions

The following functions generate IDs of groups.

dbStartGenGroup

```
dbGenStateId  
dbStartGenGroup(  
    dbCellViewId cellViewId  
);
```

The `dbStartGenGroup` function works with the `dbGenGroup` function to generate the `dbGroupId` of all groups contained in the specified cellview.

dbStartGenGroupWithDef

```
dbGenStateId  
dbStartGenGroupWithDef(  
    dbCellViewId cellViewId  
    const char *groupDefName  
);
```

Generates the groups contained in the design of the specified `groupDef`. If the specified `groupDef` cannot be found, a NULL `dbGenStateId` is returned.

dbGenGroup

```
Boolean  
dbGenGroup(  
    dbGenStateId pState, dbGroupId pGroupId  
);
```

Generates the `dbGroupId` of all groups contained in the specified cellview.

dbGenGroupWithDef

Boolean

```
dbGenGroupWithDef(  
    dbGenStateId stateId,  
    dbGroupId *groupId  
);
```

Generates all groups of the specified groupDef in the cellview.

dbStartGenGroupByName

dbGenStateId

```
dbStartGenGroupByName(  
    dbCellViewId cellViewId, String name  
);
```

Starts the generator for the dbGroupId of all groups with the specified name contained in the specified cellview.

dbGenGroupByName

Boolean

```
dbGenGroupByName(  
    dbGenStateId pState, dbGroupId pGroupId  
);
```

Generates the dbGroupId of all groups with the specified name contained in the specified cellview.

dbStartGenGroupToId

dbGenStateId

```
dbStartGenGroupToId(  
    dbGroupId groupId  
);
```

Starts the generator for the dbId of all objects contained in the specified group. If an object is a member of the group more than once, it is generated as many times as it is a member in the group. You generate, in the order you specify, IDs in an ordered group by creating, deleting, and moving group members.

dbGenGroupTold

Boolean

```
dbGenGroupToId(  
    dbGenStateId pState, dbId pId  
);
```

Generates the `dbId` of all objects contained in the specified group. If an object is a member of the group more than once, it is generated as many times as it is a member in the group. You generate, in the order you specify, IDs in an *ordered* group by creating, deleting, and moving group members.

dbStartGenGroupToGroupMem

dbGenStateId

```
dbStartGenGroupToGroupMem(  
    dbGroupId groupId  
);
```

Starts the generator for the `GroupMemId` of all members in the specified group. Each `GroupMemId` is generated once. However, if the group is a collection, the same object can have more than one `GroupMemId`. You generate, in the order you specify, `GroupMemIds` in an *ordered* group by creating, deleting, and moving group members.

dbGenGroupToGroupMem

Boolean

```
dbGenGroupToGroupMem(  
    dbGenStateId pState, dbGroupMemId pGroupMemId  
);
```

Generates the `GroupMemId` of all members in the specified group. Each `GroupMemId` is generated once. However, if the group is a collection, the same object can have more than one `GroupMemId`. You generate, in the order you specify, `GroupMemIds` in an *ordered* group by creating, deleting, and moving group members.

dbStartGenIdToGroup

dbGenStateId

```
dbStartGenIdToGroup(  
    dbId id  
);
```

Starts the generator for the `GroupId` of all groups containing the specified ID as a member. If a group is a collection and it contains multiple `GroupMemIds` for this object, the group is generated once for each `GroupMemId` that represents the object.

dbGenIdToGroup

Boolean

```
dbGenIdToGroup(  
    dbGenStateId pState, dbGroupId groupId  
);
```

Generates the `GroupId` of all groups containing the specified ID as a member. If a group is a collection and it contains multiple `GroupMemIds` for this object, the group is generated once for each `GroupMemId` that represents the object.

dbStartGenIdToGroupMem

dbGenStateId

```
dbStartGenIdToGroupMem(  
    dbId id  
);
```

Starts the generator for the `GroupMemId` of all group members corresponding to the specified ID in all groups that contain the object as a member.

dbGenIdToGroupMem

Boolean

```
dbGenIdToGroupMem(  
    dbGenStateId pState, dbGroupMemId pGroupMemId  
);
```

Generates the `GroupMemId` of all group members corresponding to the specified ID in all groups that contain the object as a member.

Pin Group Guides

Pin Group Guides provide a model for specifying the location, ordering, layers, and spacing of pins in a design. Pin group guides can be area pin group guides or edge pin group guides.

An area pin group guide represents the regions where pins should be placed. In addition to pins and pinFigs, areaBoundary objects can be used to represent the regions for placing the pins/pinFigs.

An edge pin group guide represents a region around the edge of a prBoundary where pins can be placed.

A set of associated constraints can be defined to determine the layer assignment, keepout spacing, minimum pin spacing, and pin width for the pins in a pin group guide.

Functions for Adding, Removing, and Accessing Pins

The following functions can be used for adding, removing, and accessing pins and pin figures in a pin group guide:

dbAddPinToPinGroupGuide

Boolean

```
dbAddPinToPinGroupGuide(  
    dbPinGroupGuideId pggId,  
    dbPinId pinId  
);
```

Adds a pin to the specified pin group guide.

dbSubPinFromPinGroupGuide

Boolean

```
dbSubPinFromPinGroupGuide(  
    dbPinGroupGuideId pggId,  
    dbPinId pinId  
);
```

Removes a pin from the specified pin group guide.

dbAddFigToPinGroupGuide

Boolean

```
dbAddFigToPinGroupGuide(  
    dbPinGroupGuideId pggId,  
    dbFigId figId  
);
```

Adds a via or a non-text shape to the specified pin group guide.

dbSubFigFromPinGroupGuide

Boolean

```
dbSubFigFromPinGroupGuide(  
    dbPinGroupGuideId pggId,  
    dbFigId figId  
);
```

Removes a via or a non-text shape from the specified pin group guide.

dbStartGenPinGroupGuidePins

dbGenStateId

```
dbStartGenPinGroupGuidePins(  
    dbPinGroupGuideId pinGroupGuideId  
);
```

This function starts and generates the list of pin group guides in a specified pin.

dbGenPinGroupGuideFigs

Boolean

```
dbGenPinGroupGuideFigs(  
    dbGenStateId stateId,  
    dbFigId *pinFigId  
);
```

This function generates the list of pin group guides in a specified pin.

dbDeletePinGroupGuide

Boolean

```
dbDeletePinGroupGuide(  
    dbPinGroupGuideId pggId  
);
```

This function deletes the specified pin group guide.

Functions for Accessing Pin Group Guide Attributes

The following functions can be used for accessing the pin group guide attributes, such as `otherFigAllowed`, pin group guide name, and pin group guide type:

dbFindPinGroupGuideByName

```
dbPinGroupGuideId  
dbFindPinGroupGuideByName(  
    dbCellViewId cellviewId,  
    constString name  
);
```

Returns the `dbPinGroupGuideId`, if the `pinGroupGuide` object exists in the specified cellview with the specified name.

dbGetPinGroupGuideType

```
dbPinGroupGuideType  
dbGetPinGroupGuideType(  
    dbPinGroupGuideId pggId  
);
```

Returns the `dbPinGroupGuideType` of the specified pin group guide. The possible return values are `area` and `edge`.

dbGetPinGroupGuideName

```
String  
dbGetPinGroupGuideName(  
    dbPinGroupGuideId pggId,  
);
```

Returns the name of the specified pin group guide.

dbSetPinGroupGuideName

```
Boolean  
dbSetPinGroupGuideName(  
    dbPinGroupGuideId pggId,  
    constString name  
);
```

Sets the name of the specified pin group guide.

dbGetPinGroupGuidePreservePinOrder

Boolean

```
dbGetPinGroupGuidePreservePinOrder(  
    dbPinGroupGuideId pggId  
);
```

This function returns a boolean value indicating whether the pin placement should preserve the pin ordering of the Pin Group Guide.

dbSetPinGroupGuidePreservePinOrder

Boolean

```
dbSetPinGroupGuidePreservePinOrder(  
    dbPinGroupGuideId pggId,  
    Boolean value  
);
```

This function sets the `preservePinOrder` attribute for the specified pin group guide.

dbGetPinGroupGuideEvenPinSpread

Boolean

```
dbGetPinGroupGuideEvenPinSpread(  
    dbPinGroupGuideId pggId  
);
```

This function returns a boolean value indicating whether the pin placement should be spread evenly across the pin placement region or areas. The default value is `true`.

dbSetPinGroupGuideEvenPinSpread

Boolean

```
dbSetPinGroupGuideEvenPinSpread(  
    dbPinGroupGuideId pggId,  
    Boolean value  
);
```

This function sets the `evenPinSpread` attribute for the specified pin group guide.

Functions for Specifying Pin Group Guide Constraints

The following functions can be used for specifying the `validRoutingLayers`, `layerPriorityOrder`, `keepoutSpacing`, `minSpacing`, and `minWidth` constraints for pin group guides:

dbPinGroupGuideHasValidRoutingLayers

Boolean

```
dbPinGroupGuideHasValidRoutingLayers(  
    dbPinGroupGuideId areaPGGId  
);
```

This function returns a boolean value indicating whether the `validRoutingLayers` constraint is set on the specified area pin group guide.

dbGetPinGroupGuideValidRoutingLayers

Boolean

```
dbGetPinGroupGuideValidRoutingLayers(  
    dbPinGroupGuideId areaPGGId,  
    unsigned int *numLayers,  
    dbLayer **layers  
);
```

This function returns the value of the `validRoutingLayers` constraint set on the specified area pin group guide.

dbSetPinGroupGuideValidRoutingLayers

Boolean

```
dbSetPinGroupGuideValidRoutingLayers(  
    dbPinGroupGuideId areaPGGId,  
    unsigned int numLayers,  
    dbLayer *layers  
);
```

This function sets the value of the `validRoutingLayers` constraint on the given area pin group guide.

dbUnsetPinGroupGuideValidRoutingLayers

Boolean


```
dbUnsetPinGroupGuideValidRoutingLayers(  
    dbPinGroupGuideId areaPGGId  
);
```

This function unsets the `validRoutingLayers` constraint on the specified area pin group guide.

dbPinGroupGuideHasLayerPriorityOrder

```
Boolean  
  
dbPinGroupGuideHasLayerPriorityOrder(  
    dbPinGroupGuideId areaPGGId  
);
```

This function returns a boolean value indicating that the `layerPriorityOrder` constraint is set on the specified area pin group guide.

dbGetPinGroupGuideLayerPriorityOrder

```
Boolean  
  
dbGetPinGroupGuideLayerPriorityOrder(  
    dbPinGroupGuideId areaPGGId  
);
```

This function returns the value of the `layerPriorityOrder` constraint set on the specified area pin group guide.

dbSetPinGroupGuideLayerPriorityOrder

```
Boolean  
  
dbSetPinGroupGuideLayerPriorityOrder(  
    dbPinGroupGuideId areaPGGId,  
    Boolean value  
);
```

This function sets the value of the `layerPriorityOrder` constraint on the specified area pin group guide.

dbUnsetPinGroupGuideLayerPriorityOrder

```
Boolean  
  
dbUnsetPinGroupGuideLayerPriorityOrder(  
    dbPinGroupGuideId areaPGGId  
);
```

This function unsets the `layerPriorityOrder` constraint on the specified area pin group guide.

dbPinGroupGuideHasKeepoutSpacing

Boolean

```
dbPinGroupGuideHasKeepoutSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

Returns a boolean value indicating that the `keepoutSpacing` constraint is set on the specified pin group guide.

dbGetPinGroupGuideKeepoutSpacing

dbDistance

```
dbGetPinGroupGuideKeepoutSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

Returns the `keepoutSpacing` value for the given layer on the specified pin group guide.

dbSetPinGroupGuideKeepoutSpacing

Boolean

```
dbSetPinGroupGuideKeepoutSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer,  
    dbDistance value  
);
```

This function sets the `keepoutSpacing` constraint value for the given layer on the specified pin group guide.

dbUnsetPinGroupGuideKeepoutSpacing

Boolean

```
dbUnsetPinGroupGuideKeepoutSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

This function unsets the `keepoutSpacing` constraint for the specified layer on the specified pin group guide.

dbPinGroupGuideHasMinPinSpacing

Boolean

```
dbPinGroupGuideHasMinPinSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

This function returns a boolean value indicating whether the `minSpacing` constraint is set for the given layer on specified pin group guide.

dbGetPinGroupGuideMinPinSpacing

dbDistance

```
dbGetPinGroupGuideMinPinSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer,  
);
```

This function returns minimum pin spacing value for the given layer on the specified pin group guide.

dbSetPinGroupGuideMinPinSpacing

Boolean

```
dbSetPinGroupGuideMinPinSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer,  
    dbDistance value  
);
```

This function sets the `minSpacing` constraint value for the given layer on the specified pin group guide.

dbUnsetPinGroupGuideMinPinSpacing

Boolean

```
dbUnsetPinGroupGuideMinPinSpacing(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

This function unsets the `minSpacing` constraint for the given layer on the specified pin group guide.

dbPinGroupGuideHasMinPinWidth

Boolean

```
dbPinGroupGuideHasMinPinWidth(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

This function returns a boolean value indicating whether the `minWidth` constraint is set for the given layer on the specified pin group guide.

dbGetPinGroupGuideMinPinWidth

dbDistance

```
dbGetPinGroupGuideMinPinWidth(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

This function returns the `minWidth` constraint value for the given layer on the specified pin group guide.

dbSetPinGroupGuideMinPinWidth

Boolean

```
dbSetPinGroupGuideMinPinWidth(  
    dbPinGroupGuideId pggId,  
    dbLayer layer,  
    dbDistance value  
);
```

This function sets the `minWidth` constraint value for the given layer on the given pin group guide.

dbUnsetPinGroupGuideMinPinWidth

Boolean

```
dbUnsetPinGroupGuideMinPinWidth(  
    dbPinGroupGuideId pggId,  
    dbLayer layer  
);
```

This function unsets the `minWidth` constraint for the given layer on the specified pin group guide.

Area Pin Group Guides Functions

The following functions are specific to area pin group guides:

dbCreateAreaPinGroupGuide

```
dbPinGroupGuideId  
dbCreateAreaPinGroupGuide(  
    dbCellViewId cellViewId,  
    constString name  
);
```

Creates an area pin group guide with the specified name in the given cellview.

dbAddAreaToPinGroupGuide

```
Boolean  
dbAddAreaToPinGroupGuide(  
    dbPinGroupGuideId areaPGGId,  
    dbAreaBoundaryId area  
);
```

This function adds an area boundary to the specified area pin group guide. The boundary must be in the same design as this pin group guide.

dbSubAreaFromPinGroupGuide

```
Boolean  
dbSubAreaFromPinGroupGuide(  
    dbPinGroupGuideId areaPGGId,  
    dbAreaBoundaryId area  
);
```

This function removes an area boundary from the specified pin group guide.

dbGetAreaBoundaryPinGroupGuide

```
Boolean
```

```
dbGetAreaBoundaryPinGroupGuide(  
    dbAreaBoundaryId id  
);
```

This function returns the area pin group guide associated with the specified area boundary.

dbAreaBoundaryHasValidRoutingLayers

Boolean

```
dbAreaBoundaryHasValidRoutingLayers(  
    dbAreaBoundaryId id  
);
```

This function returns a boolean value indicating that the `validRoutingLayers` constraint is set on the specified area boundary.

dbGetAreaBoundaryValidRoutingLayers

Boolean

```
dbGetAreaBoundaryValidRoutingLayers(  
    dbAreaBoundaryId id,  
    unsigned int *numLayers,  
    dbLayer **layers  
);
```

This function returns the value of the `validRoutingLayers` constraint on the specified area boundary.

dbSetAreaBoundaryValidRoutingLayers

Boolean

```
dbSetAreaBoundaryValidRoutingLayers(  
    dbAreaBoundaryId id,  
    unsigned int numLayers,  
    dbLayer *layers  
);
```

This function sets the value of the `validRoutingLayers` constraint on the specified area boundary.

dbUnsetAreaBoundaryValidRoutingLayers

Boolean

```
dbUnsetAreaBoundaryValidRoutingLayers(  
    dbAreaBoundaryId id  
);
```

This function unsets the `validRoutingLayers` constraint value on the specified area boundary.

dbAreaBoundaryHasLayerPriorityOrder

```
Boolean  
dbAreaBoundaryHasLayerPriorityOrder(  
    dbAreaBoundaryId id  
);
```

This function returns a boolean value indicating that the `layerPriorityOrder` constraint is set on the specified area boundary.

dbGetAreaBoundaryLayerPriorityOrder

```
Boolean  
dbGetAreaBoundaryLayerPriorityOrder(  
    dbAreaBoundaryId id  
);
```

This function returns the value of the `layerPriorityOrder` constraint that is set on the specified area boundary.

dbSetAreaBoundaryLayerPriorityOrder

```
Boolean  
dbSetAreaBoundaryLayerPriorityOrder(  
    dbAreaBoundaryId id,  
    Boolean value  
);
```

This function sets the value of the `layerPriorityOrder` constraint on the specified area boundary.

dbUnsetAreaBoundaryLayerPriorityOrder

```
Boolean  
dbUnsetAreaBoundaryLayerPriorityOrder(  
    dbAreaBoundaryId id  
);
```

This function deletes the `layerPriorityOrder` constraint on the specified area boundary.

dbStartGenPinGroupGuideAreas

```
dbGenStateId  
dbStartGenPinGroupGuideAreas(  
    dbPinGroupGuideId pinGroupGuideId  
);
```

This function starts and generates the list of areas in a specified pin group guide.

dbGenPinGroupGuideAreas

```
Boolean  
dbGenPinGroupGuideAreas(  
    dbGenStateId stateId,  
    dbAreaBoundaryId *areaBoundaryId  
);
```

This function generates the list of areas in the specified pin group guide.

Edge Pin Group Guide Functions

The following functions are specific to edge pin group guides:

dbCreateEdgePinGroupGuide

```
dbPinGroupGuideId  
dbCreateEdgePinGroupGuide(  
    dbCellViewId cellViewId,  
    constString name  
);
```

Creates an edge pin group guide with the specified name in the given cellview.

dbGetPinGroupGuideEdgeBoundary

```
dbPRBoundaryId  
dbGetPinGroupGuideEdgeBoundary(  
    dbPinGroupGuideId edgePGGId  
);
```

Returns the `prBoundary` associated with the specified edge pin group guide.

dbGetPinGroupGuideEdgeName

String

```
dbGetPinGroupGuideEdgeName (
    dbPinGroupGuideId edgePGGId
);
```

Returns the edge name of the specified edge pin group guide.

dbGetPinGroupGuideEdgeStartOffset

dbCoord

```
dbGetPinGroupGuideEdgeStartOffset (
    dbPinGroupGuideId edgePGGId
);
```

Returns the start offset of the edge for the specified edge pin group guide.

dbGetPinGroupGuideEdgeStopOffset

dbCoord

```
dbGetPinGroupGuideEdgeStopOffset (
    dbPinGroupGuideId edgePGGId
);
```

Returns the stop offset of the edge for the specified edge pin group guide.

dbSetPinGroupGuideEdgeStartOffset

Boolean

```
dbSetPinGroupGuideEdgeStartOffset (
    dbPinGroupGuideId edgePGGId,
    dbCoord offset
);
```

This function sets the start offset of the edge on the edge pin group guide.

dbSetPinGroupGuideEdgeStopOffset

Boolean

```
dbSetPinGroupGuideEdgeStopOffset (
    dbPinGroupGuideId edgePGGId,
    dbCoord offset
);
```

This function sets the stop offset of the edge on the edge pin group guide.

dbRemovePinGroupGuideEdge

Boolean

```
dbRemovePinGroupGuideEdge (
    dbPinGroupGuideId edgePGGId
);
```

This function removes the `prBoundary` and edge name specification from the specified edge pin group guide.

dbSetPinGroupGuideEdge

Boolean

```
dbSetPinGroupGuideEdge (
    dbPinGroupGuideId edgePGGId,
    dbPRBoundaryId prBoundaryId
);
```

This function specifies new or updates existing `prBoundary` edge information for the edge pin group guide.

Pin Group Guide Functions for Other Objects

You can use the following functions to retrieve information about objects associated with pin group guides:

dbPinHasMinPinSpacing

Boolean

```
dbPinHasMinPinSpacing (
    dbPinId pinId,
    dbLayer layer
);
```

This function returns a boolean value indicating whether the `minSpacing` constraint is set on the specified pin.

dbGetPinMinPinSpacing

dbDistance

```
dbGetPinMinPinSpacing(  
    dbPinId pinId,  
    dbLayer layer  
);
```

This function returns the value of the `minSpacing` constraint set on the specified pin.

dbSetPinMinPinSpacing

Boolean

```
dbSetPinMinPinSpacing(  
    dbPinId pinId,  
    dbLayer layer,  
    dbDistance value  
);
```

This function sets the value of the `minSpacing` constraint on the specified pin.

dbUnsetPinMinPinSpacing

Boolean

```
dbUnsetPinMinPinSpacing(  
    dbPinId pinId,  
    dbLayer layer  
);
```

This function deletes the `minSpacing` constraint on the specified pin.

dbPinHasMinPinWidth

Boolean

```
dbPinHasMinPinWidth(  
    dbPinId pinId,  
    dbLayer layer  
);
```

This function returns a boolean value indicating whether the `minWidth` constraint has been applied to the specified pin.

dbGetPinMinPinWidth

dbDistance

```
dbGetPinMinPinWidth(  
    dbPinId pinId,  
    dbLayer layer  
);
```

This function returns the `minWidth` constraint value for the specified pin.

dbSetPinMinPinWidth

Boolean

```
dbSetPinMinPinWidth(  
    dbPinId pinId,  
    dbLayer layer,  
    dbDistance value  
);
```

This function sets the value of the `minWidth` constraint on the given pin.

dbUnsetPinMinPinWidth

Boolean

```
dbUnsetPinMinPinWidth(  
    dbPinId pinId,  
    dbLayer layer  
);
```

This function unsets the `minWidth` constraint on the specified pin.

dbGetFigPinGroupGuide

dbPinGroupGuideId

```
dbGetFigPinGroupGuide(  
    dbFigId figId  
);
```

Returns the pin group guide associated with the specified figure.

dbStartGenCellViewPinGroupGuides

dbGenStateId

```
dbStartGenCellViewPinGroupGuides(  
    dbCellViewId cvId  
);
```

This function starts and generates the list of pin group guides in a specified `cellView`.

dbGenCellViewPinGroupGuides

Boolean

```
dbGenCellViewPinGroupGuides (
    dbGenStateId stateId,
    dbPinGroupGuideId *pinGroupGuideId
);
```

This function generates the list of pin group guides in the specified cellView.

dbStartGenPinToPinGroupGuides

dbGenStateId

```
dbStartGenPinToPinGroupGuides (
    dbPinId pinId
);
```

This function starts and generates the list of pin group guides in a specified pin.

dbGenPinToPinGroupGuides

Boolean

```
dbGenPinToPinGroupGuides (
    dbGenStateId stateId,
    dbPinGroupGuideId *pinGroupGuideId
);
```

This function generates the list of pin group guides in the specified pin.

dbStartGenPRBoundaryPinGroupGuides

dbGenStateId

```
dbStartGenPRBoundaryPinGroupGuides (
    dbPRBoundaryId prBoundaryId
);
```

This function starts and generates the list of pin group guides in a specified prBoundary.

dbGenPRBoundaryPinGroupGuides

Boolean

Cadence Integrators Toolkit Database Reference

CDBA User Database Extensions

```
dbGenPRBoundaryPinGroupGuides (
    dbGenStateId stateId,
    dbPinGroupGuideId *pinGroupGuideId
);
```

This function generates the list of pin group guides in the specified `prBoundary`.

CDBA Technology File Functions

This chapter includes the following topics:

- [Technology File Overview](#)
- [Administrative \(File\) Functions](#)
- [Technology File Management Functions](#)
- [Control Class Functions](#)
- [Layer Functions](#)
- [Equivalent Layer Functions](#)
- [Via Layers Functions](#)
- [Derived Layers](#)
- [Layer-Purpose \(LP\) Pair Functions](#)
- [Layout Suite L Rule Functions](#)
- [Virtuoso Layout Suite XL Functions](#)
- [Place and Route Rule Functions](#)
- [viaDef Functions](#)
- [viaHeader Functions](#)
- [Site Definition Functions](#)
- [Site Pattern Functions](#)
- [Physical Constraint Functions](#)
- [Electrical Rule Functions](#)

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

- Technology File Database Functions
- Trim Layer Functions

Technology File Overview

This section describes the technology file C functions that are administrative in nature. They do not affect the technology data of the technology file.

For more information on the technology file functions and technology file constraints, see these sections.

- [Uses for Technology File Functions](#) on page 1145
- [Accessing Technology Data](#) on page 1145
- [Technology File Constraints](#) on page 1146
- [Warnings in Functions](#) on page 1146
- [Speed of Implementation in Functions](#) on page 1147
- [Ordered Properties in Functions](#) on page 1147
- [Pointers in Technology File Functions](#) on page 1148

Uses for Technology File Functions

Use technology file C functions to access technology data. Specifically, you can

- Open and close a technology file in virtual memory
- Manage files opened in virtual memory. This includes
 - Purging virtual memory
 - Refreshing virtual memory with the disk version
 - Getting identifiers of open technology files
- Save the technology file opened in virtual memory to disk
- Attach a technology file to a Design Framework II library, cell, or cellview
- Set and get control parameters for use in the technology data

Accessing Technology Data

When accessing technology data, keep the following in mind:

- Retrieve a technology file from the cellview, not from the library.

- Most rules are no longer stored in classes. Now, rules are defined in constraint groups. See the Technology File Constraints sections below.
- All display information is contained in packets. Each layer-purpose pair has an associated packet stored as the packet name.

Note: OpenAccess does not support layer purposes. DFII applications do support them.

Technology File Constraints

The basic structure of a constraint section in the technology file is as follows.

```
constraintGroups (
  ("constraintGroupName"
    constraintType(
      ( specificConstraint )
      ( specificConstraint )
    );constraintType
  );constraintGroupName
);constraintGroups
```

Constraint information is defined in constraint groups. The `constraintGroups` section contains the following.

- constraint group name, such as `foundry`.
- constraint categories, such as `physical constraints`.
- constraint types, such as `spacings`.
- collection of constraints, such as `minSpacing`. The constraints are used as input to compilation during loading.

For example.

```
constraintGroups (
  ("foundry"
    ; physical constraints
    spacings(
      ( minSpacing Metall 0.3 )
      ( minSpacing Metall 0.4 )
    );spacings
  );foundry
);constraintGroups
```

Warnings in Functions

Some pairs or groups of functions differ only in the type and specificity of error or warning messages generated when the function returns FALSE. Functions whose names do not include `NoWarn` return more general error messages and are useful for simpler tasks.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Functions whose names do include `NoWarn` are called when you are performing a more complex task, which requires a more specific and meaningful error message.

An example pair of functions differing in this respect is `techList2LP` and `techList2LPNoWarn`. In both cases, the C code prompts you to enter a layer-purpose pair. You enter a layer-purpose pair the equivalent of `("someLayer" "somePurpose")`. The C code is passed this layer-purpose pair and calls the function to break the layer-purpose pair into smaller units of code for processing.

If `techList2LP` determines that `someLayer` does not exist, it gives you a general message such as `"layer'someLayer' does not exist"` and returns `FALSE`.

If you enter a layer-purpose pair which is used for a very specific task, such as for a third-layer metal, the C code might call the `techList2LPNoWarn` function. If the function returns `FALSE` (indicating an error occurred), you have the option of printing out a more meaningful message such as `"The third-layer metal specified by the layer-purpose pair does not exist. Try again."`

Speed of Implementation in Functions

Some pairs or sets of functions differ only in the speed and thoroughness in which they are implemented. The difference is indicated in the function names by the contrasting use of `2` or `To` in the function name. When `2` is used in a function name, the function processes information more quickly but not as thoroughly. Functions whose name includes `To` process information more slowly and thoroughly.

An pair of functions differing only in this respect is `techList2LP` and `techListToLP`.

Ordered Properties in Functions

A calling function passes in `TRUE` or `FALSE` to specify whether a particular property is an ordered property or not. If a property is ordered, a property between layer1 and layer 2 might have one value, whereas the a property with the same name between layer 2 and layer 1 might have another value.

If the property is not ordered, a property between layer 1 and layer 2 of a particular name has the same value regardless of whether layer 1 and layer 2 or layer 2 and layer 1 is referred to. The function returns a value of type `techGroupId`.

If the two-layer property is successfully created, a valid `techGroupId` is returned. If the return value is `techcNullGroupId`, an error occurred and the property was not created.

Pointers in Technology File Functions

A function updates the values indicated by the pointers. For example, the `dbGenFig` function updates a `dbFigId` pointer, `techGenLP` updates `techLayerNum` and `techPurpose` pointers (for a layer-purpose pair). The `techGenSpacingRule` updates pointers to layers, purposes, `dbPropType`, and `dbPropValue`, the latter being the spacing rule value.

See [“Getting Started with CDBA”](#) on page 85 for more information on pointers in generator functions.

See [“techGenSpacingRule”](#) on page 1520 for an example of how these pointers are used.

Attach Functions

Use attach functions to attach, or bind, a specific technology file to a Design Framework II library. You can also delete the binding.

The technology file binding is made up of the `techLibName` property, which is defined as follows:

<code>techLibName</code>	The name of the Design Framework II library containing the technology file.

Note: A technology file must reside in a Design Framework II library.

Administrative (File) Functions

Administrative functions do not affect the technology data of the technology file. Use administrative file functions to open, close, and manage files in virtual memory. The open functions provide three modes that control what remains on disk and what is loaded into virtual memory. The modes are

- Read—loads the contents of the binary technology file into virtual memory and does not allow any edits to the file
- Append—loads the contents of the binary technology file into virtual memory and does allow edits
- Write—deletes the contents of the binary technology file on disk and loads the empty file into virtual memory



Write mode does allow edits. Use this mode with extreme caution!

techCloseTechFile

Boolean

```
techCloseTechFile(  
    techFileId techfile  
);
```

Description

Closes the technology file and decrements the opencount. The technology file is not purged from virtual memory until the system needs to use the memory and the open count is equal to zero. Internally, the system maintains a count of the number of times you open and close a specific technology file. The count increments when you open and decrements when you close. When the open count is 0 and the system needs more virtual memory, it purges the technology file from virtual memory.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file.

Return Values

Boolean	If the technology file does not exist, this function returns FALSE.

Example

```
Boolean status;  
techFileId techFile;  
status = techCloseTechFile(techfile);
```

techCopyTechFile

```
techFileId  
techCopyTechFile(  
    techFileId    oldTechFile,  
    String        newLibName,  
    String        path,  
    Boolean        deleteOriginal  
);
```

Description

Copies a technology file to a new location and optionally deletes the original. This function is especially useful for separating technology data out of libraries that contain both design data and technology data.

Arguments

oldtechFile	The database (db) identifier of the old technology file.
newLibName	The name of the new library that the technology file is copied to.
path	The path of the new library.
deleteOriginal	The option to delete the original technology file, including binary and all device cellviews.

Return Values

techFileId	The database identifier of the new technology file. All technology devices are copied also. If the technology file does not exist, this function returns <code>techcNullTechFileId</code> .

Example

```
techFileId newTechId;  
newTechId = techCopyTechFile( oldTechId,  
    "newTech", "/usr/myDir/TechFiles", TRUE);
```

techDeleteTechFile

Boolean

```
techDeleteTechFile(  
    techFileId techFile  
);
```

Description

Deletes a technology file. This function is useful for deleting redundant technology data from design libraries that have been updated to share one technology library.

Arguments

techFile	The database (db) identifier of the technology file.

Return Values

Boolean	If the technology file does not exist, this function returns FALSE (an error).

Example

```
Boolean status;  
techFileId techFile;  
status = techDeleteTechFile(techFile);
```


techGetCreateTime

```
time_t  
techGetCreateTime( techFileId    techfile  
                  );
```

Returns the time at which the technology file was created.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file.

Return Values

<code>techTime</code>	The time the technology was created.

Example

```
techFileId techFile;  
techfile = techGetCreateTime();
```

techGetTechFileDdId

```
ddId  
techGetTechFileDdId(  
    techFileId techfile  
);
```

Description

Returns the design data identifier for the technology file attached to the specified Design Framework II library or cellview. If the technology file does not exist, this function returns techcNullTechFileId.

For more information about design data identifiers, refer to the [*Virtuoso Design Environment SKILL Reference*](#).

Arguments

techfile	The database (db) identifier of the technology file.

Return Values

ddId	The design data identifier of the technology file. Returns an error if the technology file does not exist.

Example

```
ddId id;  
techFileId techfile;  
id = techGetTechFileDdId(techfile);
```

techGetTimeStamp

```
techTime  
techGetTimeStamp(  
    techFileId techfile  
);
```

Description

Gets the internal time stamp of the technology file to the current time. The time stamp is an integer representing the number of seconds elapsed since 00:00:00 GMT, January 1, 1970. The time stamp represents the last time the technology file was updated.

Arguments

techfile	The database (db) identifier of the technology file.

Return Values

techTime	The current internal time stamp.

Example

```
techTime currentTime;  
techFileId techfile;  
currentTime = techGetTimeStamp( techfile);
```

techOpenDefaultTechFile;

```
techFileId  
techOpenDefaultTechFile(  
    );
```

Description

Loads the default binary technology file in virtual memory in read mode and returns the database identifier. The default technology file resides at

your_install_dir/tools/dfII/etc/cdsDefTechLib

If the default technology file is not at the expected location, this function returns an error.

Arguments

None.

Return Values

<code>techFileId</code>	The database (db) identifier of the default binary technology file.

Example

```
techFileId techFile;  
techfile = techOpenDefaultTechFile();
```

techOpenTechFile

```
techFileId  
techOpenTechFile(  
    String    libName,  
    String    path,  
    String    modeString  
);
```

Description

Loads a binary technology file in virtual memory with a specified mode and returns the database identifier of the file. If you specify `r` mode, the file can only be read. If you specify `w` mode, the contents of the disk file are deleted or truncated, and an empty technology file is loaded into virtual memory. If you specify `a` mode, the technology file is loaded in append mode, which means that you can edit the contents of the file.



If you open a technology file in write mode, the contents of the disk file are deleted. Use this function with extreme care.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>libName</code>	The library name in which to load the binary technology file.
<code>path</code>	The name of the binary technology file to open. A valid value is <code>techfile.cds</code> .
<code>modeString</code>	The mode in which to open the file. Valid values: <code>r</code> (read only), <code>w</code> (delete contents and load empty file), or <code>a</code> (append, or edit mode)

Return Values

<code>techFileId</code>	The database (<code>db</code>) identifier of the technology file loaded into virtual memory. If the library or file do not exist, this function returns <code>techcNullTechFileId</code> .

Example

```
techFileId techFile;  
String libName = "myLib";  
String techfile = "techfile.cds";  
String modeString = "a";  
techFile = techOpenTechFile(libName,techfile.cds, modeString);
```

techPurgeTechFile

Boolean

```
techPurgeTechFile(  
    techFileId techfile  
);
```

Description

Deletes the file from virtual memory, regardless of the open count.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file to be purged.

Return Values

Boolean	If the technology file is invalid (does not exist), this function returns an error (a value of FALSE is returned). If the technology file is valid, a value of TRUE is returned.

Example

```
Boolean status;  
techFileId techFile;  
status = techPurgeTechFile( techfile);
```

techRefreshTechFile

Boolean

```
techRefreshTechFile(  
    techFileId techfile  
);
```

Description

Replaces the technology file loaded in virtual memory with the binary file stored on disk. The edit mode (append, read, or write) of the file remains the same. If you refresh a technology file open in append mode, any changes you made and did not save are lost. If you refresh a technology file open in write mode, the empty disk file is loaded into virtual memory. This function is sometimes referred to as `DiscardEdits`.



All changes not saved to disk are lost when you use this function.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file loaded into virtual memory.

Return Values

Boolean	If the technology file does not exist, this function returns FALSE.

Example

```
Boolean status;  
techFileId techFile;  
status = techRefreshTechFile( techFile);
```


techReopenTechFile

```
techFileId  
techReopenTechFile(  
    techFileId  techfile,  
    String      modeString  
);
```

Description

Changes the mode of a technology file that has been opened. Use this function to upgrade the mode from read to append.



Mode changes other than from read to append are not recommended. If you change the mode from read or append to write, the contents of both the disk file and virtual memory are deleted. (This function does not let you reopen in write mode.) If you change the mode from append or write to read, the file contents are not refreshed and you are not prompted to save your changes when you exit.

Arguments

techfile	The database (db) identifier of the technology file that has been reopened.
modeString	The current mode of the technology file.

Return Values

Boolean	If the technology file does not exist, this function returns FALSE (an error).

Example

```
techFileId techFile;  
techFile = techReopenTechFile( techFile, "a");
```

techSaveTechFile

Boolean

```
techSaveTechFile(  
    techFileId techfile  
);
```

Description

Saves the specified technology file from virtual memory to the disk file (including all devices) from which it was opened.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file.

Return Values

<code>Boolean</code>	If the technology file identifier is invalid, or the system is not able to write to the directory containing the disk file and devices, this function returns FALSE.

Example

```
Boolean status;  
techFileId techFile;  
status = techSaveTechFile(techFile);
```

techSaveTechFilesInITDBGraph

Boolean

```
techSaveTechFilesInITDBGraph(  
    techFileId  techId,  
    Boolean     forceToOpen  
);
```

Description

Saves the specified technology database and its referenced technology databases from virtual memory to disk.

Arguments

techId	The identifier of the technology database to save along with all of the technology databases it references.
forceToOpen	When TRUE, any non-writable library or libraries are opened in a (append) mode and then opened with the original mode. When FALSE and the library or libraries are non-writable, issues a warning message.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

techTruncateTechFile

Boolean

```
techTruncateTechFile(  
    techFileId techfile  
);
```

Description

Deletes the contents of the technology file stored on disk. This has the same effect as opening the technology file in write mode.



Caution

This function deletes data from the disk. Use this function with extreme care.

Arguments

techfile	The database (db) identifier of the technology file whose contents will be deleted.

Return Values

Boolean	If the technology file does not exist, this function returns FALSE

Example

```
Boolean status;  
techFileId techFile;  
status = techTruncateTechFile( techFile);
```

techVerifyTechFileId

Boolean

```
techVerifyTechFileId(  
    techFileId techfile  
);
```

Description

Indicates whether the specified database identifier is for a technology file.

Arguments

techfile	The database (db) identifier of the technology file.

Return Values

Boolean	If the identifier is invalid or is not for a technology file, this function returns FALSE.

Example

```
Boolean status;  
techId techfile;  
status = techVerifyTechFileId( techfile);
```

Technology File Management Functions

The following functions pertain to technology file management.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

***FuncConflictTriggerPtr**

Boolean

```
( *FuncConflictTriggerPtr )  
    ( techConflictType      conflictType,  
      int                  numObj,  
      techId               *conflictIDs  
    );
```

Description

Prototype for application code to follow when it is necessary to execute a function when there are conflicts in incremental technology databases. The value of the pointer (**conflictIDs*) is valid only if the function returns TRUE.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>conflictType</code>	The type of conflict type. Conflict types are: <code>techLayerNameConflictType</code> <code>techLayerNumConflictType</code> <code>techDerivedLayerConflictType</code> <code>techPurposeNameConflictType</code> <code>techPurposeNumConflictType</code> <code>techSiteDefNameConflictType</code> <code>techViaDefNameConflictType</code> <code>techViaSpecConflictType</code> <code>techAnalysisLibNameConflictType</code> <code>techConstraintNameConflictType</code> <code>techConstraintGroupNameConflictType</code>
<code>numObj</code>	The number of objects that have the conflicts.
<code>*conflictIDs</code>	The pointer to the database identifiers of the objects which have conflicts.

Return Values

<code>Boolean</code>	Returns TRUE if the operation is successful. Returns FALSE otherwise.

***FuncGraphTriggerPtr**

Boolean

```
( *FuncGraphTriggerPtr )  
    ( techFileId      techId  
      );
```

Description

Prototype for application code to follow when it is necessary to execute a function to modify the technology database graph.

Arguments

techId	The identifier of the technology database for which to set up the graph modification trigger function.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

techDeleteCellViewTechLibName

Boolean

```
techDeleteCellViewTechLibName (  
    dbCellViewId cv  
);
```

Description

“Unattaches” a technology library name attached to a specified cellview.

Arguments

cv	The database (db) identifier of a specified cellview.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE if an error is detected.

Example

```
Boolean status;  
dbCellViewId cv;  
status = techDeleteCellViewTechLibName (cv);
```

techDeleteObjTechLibName

Boolean

```
techDeleteObjTechLibName (  
    ddId obj  
);
```

Description

“Unattaches” a technology library name from an object. The object now inherits the attached technology file from its parent object.

Arguments

obj	The object attached to the technology file from which you want to delete the technology library name.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE if an error is detected.

Example

```
Boolean status;  
ddId obj;  
status = techDeleteObjTechLibName(obj);
```

techHasConflict

Boolean

```
techHasConflict(  
    techFileId    techId  
);
```

Description

Checks whether or not a specific technology database has one or more conflicts or is incomplete.

Arguments

techId	The identifier of the technology database to check for conflicts and incompleteness.

Return Values

Boolean	Returns TRUE if the specified technology database has a conflict or is incomplete. Returns FALSE otherwise.

techExit

```
void  
techExit(  
    );
```

Description

Frees up memory and prepares for the exit of the process. Do not call any technology file functions afterwards.

Arguments

None.

Return Values

None.

Example

```
void  
techExit();
```

techGetCellViewDeviceTechFile

```
techFileId  
techGetCellViewDeviceTechFile(  
    dbCellViewId cv  
);
```

Description

Returns the database identifier of the technology file containing technology file devices.

Arguments

cv	The database (db) identifier of the device cellview.

Return Values

techFileId	The identifier of the technology file.

Example

```
techFileId techFile  
dbCellViewId cv;  
techFile = techGetCellViewDeviceTechFile(cv);
```

techGetCellViewTechFile

```
techFileId  
techGetCellViewTechFile(  
    dbCellViewId cv  
);
```

Description

Returns the technology file attached to a specific cellview. If it is not explicitly attached at the cellview level, this function will work its way up the hierarchy from the cellview level to the cell level, and finally, to the library level.

Arguments

cv	The database (db) identifier of any cellview.

Return Values

techFileId	The database identifier of the technology file.

Example

```
techFileId techfile;  
dbCellViewId cv;  
techFile = techGetCellViewTechFile( cv);
```

techGetCellViewTechLibName

String

```
techGetCellViewTechLibName(  
    dbCellViewId cv  
);
```

Description

Returns the name of the technology library attached to a specified cellview of dbCellViewId.

Arguments

cv	The database (db) identifier of a specific cellview.

Return Values

String	The name of the technology library attached to a specified cellview.

Example

```
String techlib;  
dbCellViewId cv;  
techlib = techGetCellViewTechLibName(cv);
```


techGetDefTechLibName

```
String  
techGetDefTechLibName(  
    String viewType  
);
```

Description

Returns the default Cadence® defined default technology file, usually `cdsDefTechLib`.

Arguments

<code>viewType</code>	Can be set by view type, but this doesn't work. Pass in <code>dbcNullString</code> as an argument.

Return Values

<code>String</code>	The name of the default defined technology file.

Example

```
String defTech;  
defTech = techGetDefTechLibName( "dbcNullString");
```

techGetObjTechFile

```
techFileId  
techGetObjTechFile(  
    ddId obj  
);
```

Description

Returns the attached technology file of `ddId`. `ddId` can be that of a library, cell, or cellview.

Arguments

<code>obj</code>	Object identifier returned by <code>ddCreateLib</code> or <code>ddGetObj</code> .

Return Values

<code>techFileId</code>	The database identifier of the technology file.

Example

```
ddId obj;  
techFileId techfile;  
techfile = techGetObjTechFile(obj);
```

techGetObjTechLibName

```
String  
techGetObjTechLibName (  
    ddId obj  
);
```

Description

Retrieves the technology library name of the technology file attached to an object.

Arguments

obj	The object attached to the technology library name.

Return Values

String	The name of the technology library.

Example

```
ddId obj;  
String name;  
name = techGetObjTechLibName(obj);
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techGetReadFileName

```
String  
techGetReadFileName(  
    techFileId techfile  
);
```

Description

Returns the path of the binary technology file (`techfile.cds`).

Arguments

<code>techfile</code>	The database (<code>db</code>) identifier of the technology file.

Return Values

<code>String</code>	The name of the path to the binary technology file (<code>techfile.cds</code>).

Example

```
String techFilePath;  
techFileId techfile;  
techFilePath = techGetReadFileName( techfile);
```

techGetTechFileMode

```
String  
techGetTechFileMode(  
    techFileId techfile  
);
```

Description

Returns the current open mode of the technology file, either "r" = read, "w" = write, or "a" = append.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file.

Return Values

<code>String</code>	The current open mode of the technology file.

Example

```
String openMode;  
techFileId techfile;  
openMode = techGetTechFileMode(techfile);
```

techGetTechFileName

```
String  
techGetTechFileName(  
    techFileId techfile  
);
```

Description

Returns the name (string value) of the of the binary technology file. This is currently always `techfile.cds`.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Values

<code>String</code>	The name of the technology file.

Example

```
String techFileName;  
techFileId techfile;  
techFileName = techGetTechFileName(techfile);
```

techGetTechLibName

```
String  
techGetTechLibName(  
    techFileId techfile  
);
```

Description

Returns the string value of the `techLib` name.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Values

<code>String</code>	The name of the technology file library.

Example

```
String libName;  
techFileId techfile;  
libName = techGetTechLibName(techfile);
```

techGetWriteFileName

```
String  
techGetWriteFileName(  
    techFileId techfile  
);
```

Description

Returns the path of the write technology file.

Arguments

<code>techfile</code>	The database (db) identifier of the technology file.

Return Values

<code>String</code>	The name of the path to the write technology file.

Example

```
techFilePath = techGetWriteFileName(techfile);
```


techIdToList

```
list  
techIdToList(  
    techId id  
);
```

Description

Creates a list containing the specified identifier of the technology file or technology file object. This function is useful for passing an identifier to the SKILL level.

Arguments

<code>id</code>	The specified identifier of the technology file or technology file object. For example, <code>id</code> can be a technology file, or it can be a layer-purpose pair.

Return Values

<code>list</code>	The list of technology files containing the specified identifier.

Example

```
list techobj;  
techId id;  
techobj = techIdToList(id);
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techInit

```
String  
techInit(  
    int      *pArgc,  
    char     *pArgv  
);
```

Description

Initializes all internal technology file data structures. This function must be called prior to any technology file functions, usually in `main()`. The arguments `*pArgc` and `*pArgv` are not used when the function is passed from `main()`.

Arguments

<code>*pArgc</code>	The argument count passed to your program's main function.
<code>*pArgv</code>	The arguments passed to your program's main function.

Return Values

<code>String</code>	The current version of the technology file software (such as version 4.4.3 or 5.0.0).

Example

```
String version;  
int argc;  
char *argv;  
version = techInit( &argc, argv);
```

techIsLPId

```
Boolean techIsLPId(  
    techLPId lpId  
);
```

Description

Determines whether the passed identifier is that of a layer-purpose pair.

Arguments

lpId	The database identifier of the passed identifier (possibly of the layer-purpose pair).

Return Values

Boolean	Returns TRUE if the identifier is for a layer-purpose pair, and FALSE if it is not.

Example

```
Boolean status;  
techLPId lpId;  
status = techIsLPId(lpId);
```

techRegConflictTrigger

Boolean

```
techRegConflictTrigger(  
    FuncConflictTriggerPtr    triggerFunc,  
    unsigned int              priority,  
    const char                *name  
);
```

Description

Registers an application's function to the conflict trigger. The value of the pointer (**name*) is valid only if the function returns TRUE.

Arguments

<code>triggerFunc</code>	The name of the function to be executed if there is any conflict in the technology database.
<code>priority</code>	The priority of the function.
<code>*name</code>	The pointer to the name of the function.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

techRegGraphModTrigger

Boolean

```
techRegGraphModTrigger(  
    FuncGraphTriggerPtr    triggerFunc,  
    unsigned int            priority,  
    const char              *name  
);
```

Description

Registers an application's function to the graph modification trigger. The value of the pointer (*name) is valid only if the function returns TRUE.

Arguments

triggerFunc	The name of the function to be executed when a graph modification is required.
priority	The priority of the function.
*name	The pointer to the name of the function.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

techSetCellViewTechLibName

Boolean

```
techSetCellViewTechLibName(  
    dbCellViewId cv, String tfName  
);
```

Description

Sets the technology library name to attach to a specified cellview.

Arguments

<code>cv</code>	The database (db) identifier of a specified cellview.
<code>tfName</code>	The name of the technology library to attach to the cellview.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE if an error is detected.

Example

```
Boolean status;  
dbCellViewId cv;  
String tlName;  
status = techSetCellViewTechLibName(cv, tlName);
```

techSetObjTechLibName

Boolean

```
techSetObjTechLibName(  
    ddId obj, String tfName  
);
```

Description

Sets the technology library name of the technology file to attach to an object.

Arguments

<code>obj</code>	The object attached to the technology library name.
<code>tfName</code>	The name of the technology file to attach to the object.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE if an error is detected.

Example

```
Boolean status;  
ddId obj;  
String tfName;  
status = techGetObjTechLibName(obj, tfName);
```

techStringToPurpose

```
techPurpose  
techStringToPurpose(  
    techFileId    techfile,  
    String        name  
)
```

Description

Returns `techPurpose` from the purpose name.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>name</code>	The purpose name.

Return Values

<code>techPurpose</code>	The value of purpose.

Example

```
techPurpose    purpose;  
techfile = techGetTechFile(ddGetObj("mylib"));  
purpose = techStringToPurpose(techfile "drawing");
```


techUnattachTechFile

Boolean

```
techUnattachTechFile(  
    ddId obj  
);
```

Description

“Unattaches” a specific technology file from an object. The object now inherits the attached technology file from its parent object.

Arguments

obj	The design data (dd) identifier. The object attached to the technology file.

Return Values

Boolean	This function returns TRUE if the operation is successful. If the identifier is invalid or is not for a technology file, This function returns FALSE (an error).

Example

```
Boolean status;  
ddId obj;  
status = techUnattachTechFile(obj);
```

techUnregConflictTrigger

Boolean

```
techUnregConflictTrigger(  
    FuncConflictTriggerPtr    triggerFunc  
);
```

Description

Unregisters the specified conflict trigger function.

Arguments

<code>triggerFunc</code>	The conflict trigger function to unregister.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

techUnregGraphModTrigger

Boolean

```
techUnregGraphModTrigger(  
    FuncGraphTriggerPtr    triggerFunc  
);
```

Description

Unregisters the specified graph modification trigger.

Arguments

<code>triggerFunc</code>	The graph modification trigger function to unregister.

Return Values

Boolean	Returns TRUE if the operation is successful. Returns FALSE otherwise.

Control Class Functions

Use the control class functions to set global parameters and permissions that affect all constraints of the technology file.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techGenAllRef

Boolean

```
techGenAllRef
    techGenStateId    state,
    string             *refLibName
);
```

Description

Generates the name of the next reference technology database in the graph from a technology database specified by `state`. The value of this pointer (`*refLibName`) is valid only if the function returns TRUE.

Arguments

<code>state</code>	The state identifier retrieved by the <code>techStartGenAllRef</code> function.
<code>*refLibName</code>	The pointer to the name of the next reference technology database in the graph.

Return Values

Boolean	Returns TRUE if the name of the next reference technology database in the graph is successfully generated. Returns FALSE otherwise.

techGenAllRefLibName

Boolean

```
techGenAllRefLibName
    techGenStateId    state,
    string             *refLibName
);
```

Description

Generates the name of the next technology library in a graph that contains technology databases referenced, both directly and indirectly (by way of the references throughout the graph), from a technology database specified by `state`. The value of this pointer (`*refLibName`) is valid only if the function returns TRUE.

Arguments

<code>state</code>	The state identifier retrieved by the <code>techStartGenAllRefLibName</code> function.
<code>*refLibName</code>	The pointer to the name of the next technology library in the graph that contains reference technology databases.

Return Values

Boolean	Returns TRUE if the name of the next technology library in the graph containing reference databases is successfully generated. Returns FALSE otherwise.

techGenParam

Boolean

```
techGenParam(  
    techGenStateId    stateId,  
    String            *pParamName,  
    techPropType      *pType,  
    techPropValue     *pValue  
);
```

Description

Generates the next parameter from a technology file specified by `stateId`. The next parameter name, the type, and the value are returned via indirection. The values of these three pointers (`*pParamName`, `*pType`, and `*pValue`) are valid only if the function returns TRUE.

Arguments

<code>stateId</code>	The state identifier retrieved by the <code>techStartGenParam</code> function.
<code>*pParamName</code>	The next parameter name.
<code>*pType</code>	The property type of the property value.
<code>*pValue</code>	The property value.

Return Value

Boolean	Returns TRUE if the parameter is successfully generated. Returns FALSE If there is an error or if there are no more parameters to generate.

Example

```
stateId = techStartGenParam(techfile);  
while techGenParam (stateId, &name, &type, &value){  
}.....  
techStopGen (stateId);
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techGenRef

Boolean

```
techGenRef
    techGenStateId    state,
    techFileId        *techId
);
```

Description

Generates the next direct reference technology database from a technology database specified by `state`. The value of this pointer (`*techId`) is valid only if the function returns TRUE.

Arguments

<code>state</code>	The state identifier retrieved by the <code>techStartGenRef</code> function.
<code>*techId</code>	The pointer to the identifier of the next reference technology database.

Return Values

Boolean	Returns TRUE if the next reference technology database is successfully generated. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techGenRefLibName

Boolean

```
techGenRefLibName
    techGenStateId    state,
    string             *refLibName
);
```

Description

Generates the name of the next technology library that contains technology databases directly referenced by a technology database specified by `state`. The value of this pointer (`*refLibName`) is valid only if the function returns TRUE.

Arguments

<code>state</code>	The state identifier retrieved by the <code>techStartGenRefLibName</code> function.
<code>*refLibName</code>	The pointer to the name of the next reference technology library.

Return Values

Boolean	Returns TRUE if the reference technology library name is successfully generated. Returns FALSE otherwise.

techGetDerivedLayerLayer1Num

```
techLayerNum  
  
techGetDerivedLayerLayer1Num(  
    techLayerId    derivedLayerId  
);
```

Description

Gets the layer number of `layer1` associated with the specified derived layer.

Arguments

<code>derivedLayerId</code>	The database id of the derived layer.

Return Values

<code>techLayerNum</code>	The layer number of <code>layer1</code> associated with the derived layer. If an error occurs when getting the <code>layer1</code> number, returns the <code>techcNullLayer</code> layer number (195).

techGetDerivedLayerLayer2Num

```
techLayerNum  
techGetDerivedLayerLayer2Num(  
    techLayerId    derivedLayerId  
);
```

Description

Gets the layer number of `layer2` associated with the specified derived layer.

Arguments

<code>derivedLayerId</code>	The database id of the derived layer.

Return Values

<code>techLayerNum</code>	The layer number of <code>layer2</code> associated with the derived layer. If an error occurs when getting the <code>layer2</code> number, returns the <code>techcNullLayer</code> layer number (195).

techGetLayerMfgResolution

Boolean

```
techGetLayerMfgResolution(  
    techFileId      techfile.  
    techLayerNum    layerNum,  
    techPurpose      purpose,  
    techPropType     *ptype,  
    techPropValue    *pvalue  
)
```

Description

Gets the manufacturing grid resolution assigned to the specified layer.

If the layer attribute is not set (is equal to 0), then the function will attempt to look up the default technology wide value and return it. If this value is not set, then it will return NULL.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>*pType</i>	Pointer to the type of the property.
<i>*pValue</i>	Pointer to the manufacturing grid resolution applied to the layer.

Return Values

Boolean	Returns TRUE if the manufacturing grid resolution assigned to the specified layer is successfully retrieved. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techGetParam

Boolean

```
techGetParam(  
    techFileId      techfile,  
    String          paramName,  
    techPropType    *pType,  
    techPropValue   *pValue  
);
```

Description

Returns the value of the specified control parameter. The parameter type and value are returned by indirection. The type and value are only valid if the function returns TRUE.

Arguments

techfile	The database identifier of the technology file.
paramName	Name of the parameter for which you want the value.
*pType	The property type of the parameter value.
*pValue	The property value.

Return Values

Boolean	Returns TRUE if the function is successful. Returns FALSE if the technology file does not exist or if the specified parameter is not defined.

Example

```
techFileId techfile;  
String name = "lambda";  
techPropType type;  
techPropValue value;  
Boolean status;  
status = TechGetParam(techfile, name, &type, &value);
```

Retrieves the value of the `lambda` control parameter in the `techfile` technology file.

techIsMfgGridResolutionSet

Boolean

```
techIsMfgGridResolutionSet(  
    techFileId      techId  
);
```

Description

Determines whether or not the manufacturing grid resolution is set in a specific technology database.

Arguments

<code>techId</code>	The identifier of the technology database.

Return Values

Boolean	Returns TRUE if the manufacturing grid resolution is set. Returns FALSE otherwise.

techMakeEffectiveLPsLocal

Boolean

```
techMakeEffectiveLPsLocal(  
    techFileId    techId  
);
```

Description

Adds all layer-purpose pairs in the graph that are currently being used for the local technology database to the local technology database.

Arguments

techId	The database identifier of the technology database.

Return Values

Boolean	Returns TRUE if the function is successful. Returns FALSE otherwise.

techPurposeDefHasDescription

Boolean

```
techPurposeDefHasDescription(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns `TRUE` when the description attribute has been set on the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

Boolean	Returns <code>TRUE</code> if the function is successful.
	Returns <code>FALSE</code> if there is an error.

Example

```
Boolean hasDescription = techPurposeDefHasDescription(userPurposeDef3);
```


techPurposeDefHasParent

Boolean

```
techPurposeDefHasParent(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns `TRUE` if the parent purpose attribute has been set on the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

Boolean	Returns <code>TRUE</code> if the function is successful.
	Returns <code>FALSE</code> if there is an error.

Example

```
Boolean hasParent = techPurposeDefHasParent(userPurposeDef3);
```

techPurposeDefHasSigType

Boolean

```
techPurposeDefHasSigType(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns `TRUE` if the signal type attribute has been set on the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

Boolean	Returns <code>TRUE</code> if the function is successful.
	Returns <code>FALSE</code> if there is an error.

Example

```
Boolean hasSigType = techPurposeDefHasSigType(userPurposeDef3);
```

techPurposeDefHasVoltageRange

Boolean

```
techPurposeDefHasVoltageRange(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns `TRUE` if the voltage range attribute has been set on the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

Boolean	Returns <code>TRUE</code> if the function is successful. Returns <code>FALSE</code> if there is an error.

Example

```
Boolean hasVoltageRange = techPurposeDefHasVoltageRange(userPurposeDef3);
```

techSetEvaluateFlag

```
void  
techSetEvaluateFlag(  
    Boolean value  
);
```

Description

If, for some reason, you want the un-evaluated parameter expression, use `techSetEvaluate (FALSE)` before calling the `Get` function of your choice. Remember to call `techSetEvaluate (TRUE)` when you are finished, or the next user calling a `Get` function will most likely be unpleasantly surprised.

Arguments to `Get` functions are returned by means of indirection. You supply pointers to objects of a given type as arguments. These pointers are updated by the `Get` function if it is successful. The pointers are invalid if the generator or function returns `FALSE`.

All `Get` functions return any parameters through the `techPropType` and `techPropValue` pointers. The parameter returned by `techPropValue` is either evaluated or raw data. Unless specifically directed, all `Get` functions evaluate parameters before return. The ASCII dumper is the only utility or application that needs the unevaluated expression.

Arguments

<code>value</code>	Sets the value to <code>TRUE</code> or <code>FALSE</code> .

Return Value

<code>void</code>	No return value.

Example

```
_____ = techSetEvaluateFlag(value);
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techStartGenAllRefLibName

```
techGenStateId  
techStartGenAllRefLibName  
    techFileId      techId  
    );
```

Description

Initializes the generator function to generate the names of all of the libraries in a graph that contain technology databases referenced, both directly and indirectly (by way of the references throughout the graph), by a specific technology database.

Arguments

<code>techId</code>	The identifier of the technology database from which you retrieve all of the names of libraries that contain reference databases in the graph.

Return Values

<code>techGenStateId</code>	Returns the generator state identifier which must be passed to <code>techGenAllRefLibName</code> .

techSetLayerMfgResolution

Boolean

```
techSetLayerMfgResolution(  
    techFileId      techfile.  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    type,  
    techPropValue   value  
)
```

Description

Sets the manufacturing grid resolution assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>type</i>	The type of the property.
<i>value</i>	The manufacturing grid resolution applied to the layer.

Return Values

Boolean	Returns TRUE if the manufacturing grid resolution assigned to the specified layer is successfully set. Returns FALSE otherwise.

techSetParam

Boolean

```
techSetParam(  
    techFileId      techfile,  
    String          paramName,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Updates the value of the specified control parameter. If the parameter does not exist, the function creates it.

Arguments

techfile	The database identifier of the technology file.
paramName	Name of the parameter for which you want to set the value.
type	The property type of the parameter value.
value	The parameter value.

Return Values

Boolean	Returns TRUE if the function is successful.
	Returns FALSE if there is an error.

Example

```
techFileId techfile;  
String name = "lambda";  
techPropType type;  
techPropValue value;  
Boolean status;  
type = dbcIntType;  
value.aInt = 1;  
status = techSetParam(techfile, name, type, value);
```

Updates the value of the `lambda` control parameter in the `techfile` technology file.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techSetRefs

Boolean

```
techSetRefs (
    techFileId      techId,
    int             numrefs,
    techFileId      *refs
);
```

Description

Sets a list of technology databases to be directly referenced by a specific technology database. The value of the pointer (`*refs`) is valid only if the function returns TRUE.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>numrefs</code>	The number of technology databases the specified technology database directly references.
<code>*refs</code>	The pointer to the list of reference technology databases. The number of technology databases must match the number specified in the <code>numrefs</code> argument.

Return Values

Boolean	Returns TRUE if the list of reference technology databases is successfully set. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techStartGenAllRef

```
techGenStateId  
techStartGenAllRef  
    techFileId      techId  
    );
```

Description

Initializes the generator function to generate all of the technology databases in a graph that are referenced, both directly and indirectly (by way of the references throughout the graph), by a specific technology database.

Arguments

<code>techId</code>	The identifier of the technology database from which you retrieve all of the reference databases in the graph.

Return Values

<code>techGenStateId</code>	Returns the generator state identifier which must be passed to <code>techGenAllRef</code> .

techStartGenParam

```
techGenStateId  
techStartGenParam(  
    techFileId techfile  
);
```

Description

Initializes the generator function to generate parameters from a specific technology file. The `techParams` constraints in the Controls section of the technology file contain the parameters.

Arguments

<code>techfile</code>	The identifier of the technology file from which you retrieve the parameters.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier which must be passed to <code>techGenParam</code> .

Example

```
stateId = techStartGenParam(techfile);  
while (techGenParam (stateId, &paramName, &type, &value){  
}.....  
techStopGen (stateId);
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techStartGenRef

```
techGenStateId
techStartGenRef
    techFileId        techId
    );
```

Description

Initializes the generator function to generate technology databases directly referenced by a specific technology database. The `refTechLibs` subsection of the `controls` section of a technology file lists the direct reference technology databases.

Arguments

<code>techId</code>	The identifier of the technology database from which you retrieve direct reference technology databases.

Return Values

<code>techGenStateId</code>	Returns the generator state identifier which must be passed to <code>techGenRef</code> .

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techStartGenRefLibName

```
techGenStateId  
techStartGenRefLibName  
    techFileId      techId  
    );
```

Description

Initializes the generator function to generate the names of the libraries that contain the technology databases directly referenced by a specific technology database. The `refTechLibs` subsection in the `controls` section of a technology file lists the direct reference technology databases.

Arguments

<code>techId</code>	The identifier of the technology database from which you retrieve the names of the libraries that contain technology databases it directly references.

Return Values

<code>techGenStateId</code>	Returns the generator state identifier which must be passed to <code>techGenRefLibName</code> .

techStopRuleGen

```
void  
techStopRuleGen(  
    techRuleGenStateId stateId  
);
```

Description

Determines whether to stop generating the technology file constraints. The `techStopRuleGen` function should be used for or with the following functions:

```
techStartGenElectricalRule  
techStartGenSpacingRule
```

All other functions beginning with `techStartGen` are terminated with `techStopGen`.

Arguments

<code>stateId</code>	The state identifier used to determine whether to stop generating the constraint.

Return Value

<code>void</code>	No return value. The technology file constraints function stops generating when no further state identifier is found.

Example

```
techStopRuleGen(stateId);
```

techUnsetMfgGridResolution

Boolean

```
techUnsetMfgGridResolution(  
    techFileId      techId  
);
```

Description

Unsets the manufacturing grid resolution assignment (`mfgGridResolution` in controls) from a specific technology database.

Arguments

<code>techId</code>	The identifier of the technology database.

Return Values

Boolean	Returns TRUE if the manufacturing grid resolution is successfully removed. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techUnsetRefs

Boolean

```
techUnsetRefs(  
    techFileId      techId  
);
```

Description

Unsets all technology databases referenced by a specific technology database (refTechLibs in controls).

Arguments

techId	The identifier of the technology database.

Return Values

Boolean	Returns TRUE if the list of reference technology databases is successfully unset. Returns FALSE otherwise.

Layer Functions

The C-functions described in this chapter assign and access layer rules in the technology file. The layer rules describe layers as well as set physical and electrical limits.

Rules for Layers and Layer-Purpose Pairs

All C-level functions accept or return a `techLayerNum` and `techPurpose` argument when specifying layers.

Precedence in Retrieving the Value of a Rule

All rules that have layers as parameters can specify either a `techLayer` argument (only the layer name, such as "metal1") or a layer-purpose pair list (such as ("metal1" "drawing")).

You can specify rules for both a layer (indicated by the `techLayer` argument) and a specific layer-purpose pair which refers to that `techLayer` argument. When retrieving the value of a rule, the search begins at the lowest level specified, and layer-purpose pairs take precedence.

If you use C-functions that require a layer number and a purpose as arguments, and if the `techPurpose` argument is set to `techcAllPurpose`, you must use the `techLayer` argument, not a specific `techLPId` argument. You are effectively specifying only the layer for all layer-purpose pairs containing the layer number (`techLayerNum`).

If the optional layer specified is not available or applicable, the `techLayerNum` is set to `techcNullLayer`.

When specifying layers as rule parameters, you can use layer names or layer-purpose pairs. However, some applications only understand layer names with an implied purpose.

Layer Number and Purpose Number Arguments in Generator Functions

In generator functions that require a `techLayerNum` and `techPurpose` argument, the generator updates the values of these variables, if the function returns a value of TRUE meaning generation is successful. The values are invalid if the generator function returns a value of FALSE.

Layer Number and Purpose Number in Set and Get Functions

For `Set` functions, pass the `techLayerNum` and `techPurpose` arguments directly. For `Get` functions, including either `Get` or generator functions, the layer and purpose are returned by indirection, such as when you pass in a pointer to objects of type `techLayerNum` and `techPurpose`.

Equivalent Layers

In a set of equivalent layers, all layers in the set are treated in an equivalent manner. For example, in a set of three layers, the layer-purpose pair for the first, second, and third layers are equivalent.

The following sequence occurs when generating rules for a set of equivalent layers:

1. A `StartGen` function retrieves the rule.
2. The first `Gen` function generates the first rule.
3. A second `Gen` function generates each layer within that particular rule.
4. The first `Gen` function retrieves the next rule.
5. The process is repeated over again, until the generator functions have processed both all rules and layers within each rule.

Stream Layer Rules

Stream is the industry standard format for physical layout interchange—for importing and exporting geometric data. You can use this standard to export (stream out) data from Design Framework II (DFII) and then import (stream in) the data to the format of another EDA vendor, or you can do the reverse.

GDSII (Graphic Design Station II) is the official vendor and database stream from which DFII stream layer rules are derived. Every geometry in GDSII is associated with a layer number and a data type number, in the same way that every geometry in DFII is associated with a layer-purpose pair. A GDSII data type is equivalent to a DFII purpose in a layer-purpose pair.

The GDSII database consists of 64 layers (numbered 0 through 63) and 64 data types (also numbered 0-63). These layers and data types correspond very closely to the DFII `techLayerNum` (layer number) and `techPurpose` (data type) arguments in `streamLayer` rule functions. In DFII, however, you can have 255 layers and 255 purposes. There are more DFII layer-purpose pairs than GDSII layer-data type combinations because some DFII layer-purpose pairs, such as markers, grids, and highlights, should not be streamed out.

Because a DFII technology file can have many more layer-purpose pairs than GDSII can have layer-data type combinations, there cannot be a one-to-one correspondence between the two layout systems. To solve this problem, a DFII stream layer rule is used to specify the proper layer-purpose pair mapping to the smaller subset of GDSII layers and data types when a stream file is generated. These rules help define which layer-purpose pairs should be translated, and as what they should be translated. For example, a stream layer rule can map which layer number and stream layer number correspond, or which purpose number and stream number correspond.

Function Performance

The `techLayers` rules in the technology file can be identified by name or number. Many pairs of functions can be considered equivalent in the result they achieve. They differ only in the speed of processing.

For example, the function `techGetLayerNameByNumber` on page 1253 uses the layer number, while an equivalent function, `techGetLayerNum` on page 1252 uses the layer name. Functions that use the `layerNumber` argument deliver better performance than those that use the `layerName`.

Initially, you may have only the name of a layer, but need to do a lot of processing of the layer. For best performance, get the number of the layer first. Then use the get and set functions that accept the layer number.

Equivalent Layer Functions

The following functions access the `equivalentLayers` construct in the `layerRules` section of the technology file.

techStartGenEquivLayerRule

```
techGenStateId  
techStartGenEquivLayerRule(  
    techFileId techfile  
);
```

Description

Starts equivalent layer generation. First generate the rules within the `equivalentLayers` construct, then generate the equivalent layers. This function This function works in conjunction with the `techGenEquivLayerRule` function.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used in the <code>techGenEquivLayerRule</code> function.

Example

```
techGenStateId ruleStateId, stateId;  
techClassId    ruleId;  
techLayerNum   layerNum;  
techPurpose    purpose;  
  
stateId = techStartGenEquivLayerRule(techfile);  
##Get the next set (rule) of equivalent layers##  
while (techGenEquivLayerRule (stateId, &ruleId)){  
    ruleStateId = techStartGenEquivLayer (ruleId);  
    ###Get the next equivalent layer in the current set (rule) ###  
    while (techGenEquivLayer (ruleStateId, &layerNum, &purpose)){  
        ###All layer-purpose pairs within this generator are equivalent##
```

techGenEquivLayerRule

Boolean

```
techGenEquivLayerRule(  
    techGenStateId  stateId,  
    techClassId     *pRuleId  
);
```

Description

Generates equivalent layers. First generate the rules within the equivalentLayers construct, then generate the equivalent layers. This function works in conjunction with the techStartGenEquivLayerRule function.

Arguments

stateId	The generator state identifier returned from startGenEquivLayerRule.
*pRuleId	The id number of the next equivalent layer rule.

Return Value

Boolean	Returns TRUE if the function is successful and *pRuleId is valid. Returns FALSE when the generator is finished.

techStartGenEquivLayer

```
techGenStateId  
techStartGenEquivLayer(  
    techClassId, ruleId  
);
```

Description

Starts the generator for the `equivalentLayers` construct in the `layerRules` section of the technology file. This function works in conjunction with the `techGenEquivLayer` function.

An ASCII representation of `equivalentLayers` is as follows, where `ruleId` identifies a particular set of layers indicated within parentheses.

```
equivalent Layers (  
    (set of equivalent layers)  
    (another set of equivalent layers)  
    (...)  
)
```

`StartGenEquivLayer` and `GenEquivLayer` are used to generate each layer in this `ruleId`, one at a time.

Arguments

<code>ruleId</code>	The identifier returned from <code>techGenEquivLayerRule</code> .

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used when generating the current set of equivalent layers.

techGenEquivLayer

Boolean

```
techGenEquivLayer(  
    techGenStateId  stateId,  
    techLayerNum    *pLayerNum,  
    techPurpose     *pPurpose  
);
```

Description

Generates the next equivalent layer in the `equivalentLayers` construct in the `layerRules` section of the technology file. Both the pointers `*pLayerNum` and `*pPurpose` are returned via indirection.

This function works in conjunction with the `techStartGenEquivLayer` function.

Arguments

<code>stateId</code>	The generator state identifier returned from <code>startGenEquivLayer</code> .
<code>*pLayerNum</code>	The layer number of the next equivalent layer.
<code>*pPurpose</code>	The purpose number of the next equivalent layer.

Return Value

Boolean	Returns TRUE if generation is successful and <code>*pLayerNum</code> and <code>*pPurpose</code> are valid. Returns FALSE when the generator finished.

techSetEquivLayer

Boolean

```
techSetEquivLayer(  
    techFileId    techfile,  
    int           numberOfLayers,  
    techLayerNum  *pLayerNum,  
    techPurpose   *pPurpose  
);
```

Description

Appends the specified set of equivalent layers to the `equivalentLayers` construct in the `layerRules` section of the technology file. The layers represent the same kind of material.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>numberOfLayers</code>	The size of the array of (the number of) layer numbers and purposes supplied in the <code>*pLayerNum</code> and <code>*pPurpose</code> arguments.
<code>*pLayerNum</code>	The array of layer numbers.
<code>*pPurpose</code>	The array of purposes.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

Example

```
techFileId  techfile;  
int         numberOfLayers = 5;  
techLayerNum layernum[5];  
techPurpose  purpose[5];  
Boolean      status;  
  
status = techSetEquivLayer(techfile, numberOfLayers, layernum, purpose);
```

Via Layers Functions

The following functions access the `viaDefs` section of the technology file.

techStartGenViaLayer

```
techGenStateId  
techStartGenViaLayer(  
    techFileId techfile  
);
```

Description

Starts generating via layers as defined in the technology file. Via layers are included in the `viaDefs` section of the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the state identifier of the generator.

techGenViaLayer

Boolean

```
techGenViaLayer(  
    techGenStateId  stateId,  
    techLayerNum    *pLayer1Num,  
    techPurpose     *pPurpose1,  
    techLayerNum    *pLayer2Num,  
    techPurpose     *pPurpose2,  
    techLayerNum    *pLayer3Num,  
    techPurpose     *pPurpose3  
);
```

Description

Generates the next set of via layers. Via layers are included in the `viaDefs` section of the technology file.

Arguments

<code>stateId</code>	The generator state identifier returned from <code>startGenViaLayer</code> .
<code>*pLayer1Num</code>	The layer number of the conductor layer-purpose pair.
<code>*pPurpose1</code>	The purpose number of the conductor layer-purpose pair.
<code>*pLayer2Num</code>	The layer number of the via layer-purpose pair.
<code>*pPurpose2</code>	The purpose number of the via layer-purpose pair.
<code>*pLayer3Num</code>	The layer number of the conductor2 layer-purpose pair.
<code>*pPurpose3</code>	The purpose number of the conductor2 layer-purpose pair.

Return Value

Boolean	Returns TRUE when all the arguments returned are valid. Returns FALSE if an error occurs or the generator is finished.

techSetViaLayer

Boolean

```
techSetViaLayer(  
    techFileId      techfile,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techLayerNum    layer3Num,  
    techPurpose     purpose3  
);
```

Description

Appends the specified set of via layers to the `viaDefs` section of the technology file. Via layers refer to layers that conduct between two other layers.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layer1Num</code>	The layer number of the conductor1 layer-purpose pair.
<code>purpose1</code>	The purpose number of the conductor1 layer-purpose pair.
<code>layer2Num</code>	The layer number of the via layer-purpose pair.
<code>purpose2</code>	The purpose number of the via layer-purpose pair.
<code>layer3Num</code>	The layer number of the conductor2 layer-purpose pair.
<code>purpose3</code>	The purpose number of the conductor2 layer-purpose pair.

Return Value

Boolean	Returns FALSE if the technology file does not exist or if any of the specified layers do not exist.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techIsViaLayer

Boolean

```
techIsViaLayer(  
    techFileId      techfile,  
    techLayerNum    viaLayerNum,  
    techPurpose     viaPurpose  
);
```

Description

Indicates whether the specified layer is defined as a via layer or the middle layer of a via in the `viaDefs` section of the specified technology file. Via refer to layers that conduct between two other layers.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>viaLayerNum</code>	The layer number of the layer-purpose pair that is querying if it is a via layer.
<code>viaPurpose</code>	The purpose number of the layer-purpose pair that is querying if it is a via layer.

Return Value

Boolean	Returns FALSE if the technology file does not exist or the specified layer is not a via layer.

Derived Layers

A derived layer is the result of combining two layers with a special operation. In the case of `IxNoOverlapLayers` rule mapping, the derived layer indicates that one layer cannot overlap another layer. It takes two layers to define a derived layer for a `IxNoOverlapLayers` rule, so the operation is AND.

A derived layer has a name and a number. When converting `IxNoOverlapLayers` rules, the function automatically generates a derived layer using the following convention.

- Derived layer name is `noOverlapLayer` plus a number. The number starts at 1 and increments with each new derived layer from the `techDerivedLayers` construct in the `layerDefinitions` section of the technology file. The name is not reused.
- Derived layer number starts with the first unused number after 10000. This allows for user defined physical layers in the techfile. The number increments with each new derived layer from the `techDerivedLayers` construct in the `layerDefinitions` section of the technology file. The number is not reused.

Layer API functions also apply to derived layers. Derived layers do have their own generator, however. The techfile layer generator generates physical layers only.

The following object type enum applies to derived layers.

```
enum techType {  
    ...  
    techcPhysicalLayerType      = 3.  
    techcDerivedLayerType       = 4.  
    ...  
}
```

The following enum defines the value of the operation that creates a derived layer.

```
typedef enum techLayerOpEnum (  
    techcUnknownLayerOp        = 0,  
    techcAndLayerOp             = 1,  
    techcOrLayerOp              = 2,  
    techcNotLayerOp             = 3,  
    techcXorLayerOp             = 4,  
    techcTouchingLayerOp        = 5  
) techLayerOp;
```

The following are corresponding strings.

```
static CONST String techvLayerOpString [] = {  
    "unknown",  
    "AND",  
    ...  
}
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

```
"OR",  
"NOT",  
"XOR",  
"touching"  
};
```

techGetNumTechLayers

```
int  
techGetNumTechLayers(  
    techFileId    techfile  
);
```

Description

Gets the number of layers in the design.

Arguments

techfile	The database identifier of the technology file.

Return Values

int	Returns the number of layers in the design.

techIsLayerId

Boolean

```
techIsLayerId(  
    techLayerId    layerId  
);
```

Description

Validates that the layer is in the techfile.

Arguments

layerId	The database identifier of the layer.

Return Values

Boolean	Returns TRUE if the layer is in the techfile. Returns FALSE otherwise.

techGetTechLayerIdByNumber

```
techLayerId  
techGetTechLayerIdByNumber(  
    techFileId    techfile,  
    techLayerNum  layerNumber  
);
```

Description

Retrieves the layer identifier based on its number.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNumber</code>	The layer number.

Return Values

<code>techLayerId</code>	Returns the identifier of the layer associated with the <i>layerNumber</i> .

techGetTechLayerNumber

```
techLayerNum  
techGetTechLayerNumber(  
    techLayerId    layerId  
);
```

Description

Gets the layer number of the specified layerId.

Arguments

layerId	The layer identifier.

Return Values

techLayerNum	Returns the number of the layer associated with the <i>layerId</i> .

techGetTechLayerName

String

```
techGetTechLayerName(  
    techLayerId    layerId  
);
```

Description

Gets the layer name of the specified layerId.

Arguments

layerId	The layer identifier.

Return Values

String	Returns the name of the layer associated with the <i>layerId</i> .

techIsPhysicalLayerId

Boolean

```
techIsPhysicalLayerId(  
    techLayerId    layerId  
);
```

Description

Determines if the *layerId* is a physical layer.

Arguments

<i>layerId</i>	The database identifier of the layer.

Return Values

Boolean	Returns TRUE if the layer is a physical layer. Returns FALSE otherwise.

techIsDerivedLayerId

Boolean

```
techIsDerivedLayerId(  
    techLayerId    layerId  
);
```

Description

Determines if the layerId is a derived layer.

Arguments

layerId	The database identifier of the layer.

Return Values

Boolean	Returns TRUE if the layer is a derived layer. Returns FALSE otherwise.

techCreateDerivedLayer

```
techLayerId  
techCreateDerivedLayer(  
    techFileId      techfile,  
    techLayerNum    derivedLayerNum,  
    cppConstString  name,  
    techLayerNum    layer1Num,  
    techLayerNum    layer2Num,  
    techLayerOp     op  
)
```

Description

Creates a derived layer. Both *layer1Num* and *layer2Num* must be numbers of valid layers.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>derivedLayerNum</i>	The number of the new derived layer.
<i>name</i>	The name of the derived layer.
<i>layer1Num</i>	The number of the first layer in the derived layer.
<i>layer2Num</i>	The number of the second layer in the derived layer.
<i>op</i>	The enum value that defines the operation of the two layers.

Return Values

<i>techLayerId</i>	Returns the <i>techLayerId</i> of the new derived layer. Returns a NULL id and a warning message if the new derived layer number, name, and attributes match an existing layer id.

techStartGenDerivedLayer

```
techGenStateId  
techStartGenDerivedLayer(  
    techFileId      techfile  
);
```

Description

Starts generating derived layers in the techfile. Used exclusively for derived layers.

Arguments

<code>techfile</code>	The techfile from which to start generating derived layers.

Return Values

<code>techGenStateId</code>	Returns the generation state identifier.

techGenDerivedLayer

Boolean

```
techGenDerivedLayer(  
    techGenStateId    stateId,  
    techLayerId       *derivedLayerId  
);
```

Description

Generates derived layers in the techfile. Used exclusively for derived layers.

Arguments

stateId	The generator state identifier returned from StartGenDerivedLayer.
*derivedLayerId	The derived layer being generated.

Return Values

Boolean	Returns TRUE if the derived layer is successfully generated.

techGetDerivedTechLayer

```
techLayerId
techGetDerivedTechLayer (
    techFileId      techfile,
    techLayerNum    layer1Num,
    techLayerNum    layer2Num,
    techLayerOp     op
)
```

Description

Finds the derived layer using the relationship of the two specified layers.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layer1Num</i>	The number of the first layer in the derived layer.
<i>layer2Num</i>	The number of the second layer in the derived layer.
<i>op</i>	The enum value that defines the operation of the two layers.

Return Values

<i>techLayerId</i>	Returns the <i>techLayerId</i> of the derived layer.

techGetDerivedLayerNum

```
techLayerNum  
techGetDerivedLayerNum(  
    techFileId      techfile,  
    techLayerNum    layer1Num,  
    techLayerNum    layer2Num,  
    techLayerOp     op  
)
```

Description

Finds the derived layer using the relationship of the two specified layers.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layer1Num</i>	The number of the first layer in the derived layer.
<i>layer2Num</i>	The number of the second layer in the derived layer.
<i>op</i>	The enum value that defines the operation of the two layers.

Return Values

<i>techLayerNum</i>	Returns the number of the derived layer.

techGetDerivedTechLayerSpec

Boolean

```
techGetDerivedTechLayerSpec (  
    techLayerId        derivedLayerId,  
    techLayerNum       *layer1Num,  
    techLayerNum       *layer2Num,  
    techLayerOp        *op  
)
```

Description

Gets layer1, layer2, and the operation between the layers of the specified derived layer id.

Arguments

<i>derivedLayerId</i>	The database identifier of the derived layer.
<i>*layer1Num</i>	The number of the first layer in the derived layer.
<i>*layer2Num</i>	The number of the second layer in the derived layer.
<i>*op</i>	The enum value that defines the operation of the two layers.

Return Values

<i>Boolean</i>	Returns TRUE if the layers and operation are successfully retrieved.

techGetDerivedLayerSpecByNumber

Boolean

```
techGetDerivedLayerSpecByNumber (
    techFileId      techfile,
    techLayerNum    number,
    techLayerNum    *layer1Num,
    techLayerNum    *layer2Num,
    techLayerOp     *op
)
```

Description

Gets layer1, layer2, and the operation between the layers of the specified derived layer by number.

Arguments

<i>techfile</i>	The techfile.
<i>number</i>	The layer number.
<i>*layer1Num</i>	The number of the first layer in the derived layer.
<i>*layer2Num</i>	The number of the second layer in the derived layer.
<i>*op</i>	The enum value that defines the operation of the two layers.

Return Values

Boolean	Returns TRUE if the layers and operation are successfully retrieved.

techGetLayerNum

```
techLayerNum  
techGetLayerNum(  
    techFileId      techfile,  
    constString     layerName  
);
```

Description

Returns the layer number of the specified layer name defined in the `techLayers` section of the technology file.

The `techLayers` section is located in the `layerDefinitions` section of the technology file. The `techLayers` define the layers that can be matched with a purpose to create layer-purpose pairs.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerName</code>	The specified layer name defined in the <code>techLayers</code> section of the technology file.

Return Values

<code>techLayerNum</code>	Returns the layer number of the specified layer name. Returns <code>techcNullLayer</code> if there is an error, such as if the technology file does not exist or if the layer is not defined.

techGetLayerNameByNumber

String

```
techGetLayerNameByNumber(  
    techFileId    techfile,  
    techLayerNum  layerNumber  
);
```

Description

Returns the layer name of the specified layer number defined in the `techLayers` section of the technology file.

The `techLayers` section is located in the `layerDefinitions` section of the technology file. The `techLayers` define the layers that can be matched with a purpose to create layer-purpose pairs.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNumber</code>	The layer number.

Return Values

<code>String</code>	Returns the name of the layer associated with the supplied number. Returns <code>dbcNullString</code> if there is an error.

techGetLayerProp

```
techPropId  
techGetLayerProp(  
    techFileId      techfile,  
    String          layerName,  
    String          propName,  
    techPropType    *type,  
    techPropValue   *value  
);
```

Description

Returns the value of the specified layer property from the `techLayerProperties` rules in the technology file. The function uses the layer name to determine the property.

The `techLayerProperties` rules are located in the `layerDefinitions` section of the technology file. The `techLayerProperties` rules contain user-defined properties for specific layer-purpose objects. User-defined properties contain company or site specific information to include on the layers in the design.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerName</code>	The name of the layer that you get the property from.
<code>propName</code>	The property that you want the value of.
<code>*type</code>	The property type. This argument is returned by indirection.
<code>*value</code>	The property value. This argument is returned by indirection.

Return Values

<code>techPropId</code>	Returns the value of the layer property. Returns <code>dbcNullPropId</code> if the technology file or property does not exist.

techGetLayerPropByNumber

```
techPropId  
techGetLayerPropByNumber(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    String          name,  
    techPropType    *type,  
    techPropValue   *value  
);
```

Description

Returns the value of the specified layer property from the `techLayerProperties` rules in the technology file. The function uses the layer number to determine the property.

The `techLayerProperties` rules are located in the `layerDefinitions` section of the technology file. The `techLayerProperties` rules contain user-defined properties for specific layer-purpose objects. User-defined properties contain company or site specific information to include on the layers in the design.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The number of the layer that you get the property from.
<code>name</code>	The property name in question.
<code>*type</code>	The property type, returned by indirection.
<code>*value</code>	The property value, returned by indirection.

Return Values

<code>techPropId</code>	Returns the value of the layer property. Returns <code>dbcNullPropId</code> if the technology file or property does not exist

techLayerIsReserved

Boolean

```
techLayerIsReserved(  
    techLayerNum  layer  
);
```

Description

Determines whether or not a specified layer is a Cadence® system (or Cadence reserved) layer. Reserved layers have layer numbers from 128 through 255.

Arguments

layer	The layer number of the layer being queried.

Return Values

Boolean	Returns TRUE if the layer number is from 128 through 255. Returns FALSE if the layer number is from 0 through 127.

techGetLayerAbbrev

```
String  
techGetLayerAbbrev(  
    techFileId    techfile,  
    techLayerNum  layerNumber  
);
```

Description

Gets the abbreviation of the specified layer in the `techLayers` section of the technology file.

The `techLayers` section defines the layers that can be matched with a purpose to create layer-purpose pairs. The layer to update is specified by layer name or number. The `techLayers` section is located in the `layerDefinitions` section of the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNumber</code>	The number of the layer whose abbreviation you want to update.

Return Values

<code>String</code>	Returns the abbreviation of the layer. Returns <code>unknown</code> for derived layers.

techSetLayerAbbrev

Boolean

```
techSetLayerAbbrev(  
    techFileId    techfile,  
    techLayerNum  layerNumber,  
    constString   abbrev  
);
```

Description

Updates the abbreviation of the specified layer in the `techLayers` section of the technology file.

Some applications truncate layer names when they display layers in lists or forms. If the layer name is not unique when truncated to seven characters, specify an acceptable abbreviation for the applications to use instead.

The `techLayers` section defines the layers that can be matched with a purpose to create layer-purpose pairs. The layer to update is specified by layer name or number. The `techLayers` section is located in the `layerDefinitions` section of the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNumber</code>	The number of the layer whose abbreviation you want to update.
<code>abbrev</code>	The new abbreviation for the specified layer.

Return Values

Boolean	Returns FALSE for derived layers.

Example

```
Boolean status;  
techFileId techfile;  
techLayerNum layerNumber;  
String abbrev = "met1";  
status = techSetLayerAbbrev (techfile, layerNumber, abbrev);
```

techSetLayerName

Boolean

```
techSetLayerName (  
    techFileId      techfile,  
    techLayerNum    layerNumber,  
    constString     layerName  
);
```

Description

Updates the name of the specified layer in the `techLayers` section of the technology file.

Some applications truncate layer names when they display layers in lists or forms. If the layer name is not unique when truncated to seven characters, specify an acceptable abbreviation for the applications to use instead.

The `techLayers` section defines the layers that can be matched with a purpose to create layer-purpose pairs. The layer to update is specified by layer name or number. The `techLayers` section is located in the `layerDefinitions` section of the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNumber</code>	The layer number of the specified layer name.
<code>layerName</code>	The name of the specified layer in the technology file.

Return Values

Boolean	Returns FALSE if the layer is not defined or the technology file does not exist.

Example

```
Boolean status;  
techFileId techfile;  
techLayerNum layerNumber = 10;  
String layerName = "metall";  
status = techSetLayerName (techfile, layerNumber, layerName);
```

techSetLayerMfgResolution

Boolean

```
techSetLayerMfgResolution(  
    techFileId      techfile.  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    type,  
    techPropValue   value  
)
```

Description

Sets the manufacturing grid resolution assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>type</i>	The type of the property.
<i>value</i>	The manufacturing grid resolution applied to the layer.

Return Values

Boolean	Returns TRUE if the manufacturing grid resolution assigned to the specified layer is successfully set. Returns FALSE otherwise.

techGetLayerMfgResolution

Boolean

```
techGetLayerMfgResolution(  
    techFileId      techfile.  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *ptype,  
    techPropValue   *pvalue  
)
```

Description

Gets the manufacturing grid resolution assigned to the specified layer.

If the layer attribute is not set (is equal to 0), then the function will attempt to look up the default technology wide value and return it. If this value is not set, then it will return NULL.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>*pType</i>	Pointer to the type of the property.
<i>*pValue</i>	Pointer to the manufacturing grid resolution applied to the layer.

Return Values

Boolean	Returns TRUE if the manufacturing grid resolution assigned to the specified layer is successfully retrieved. Returns FALSE otherwise.

techGetLayerMaskNumber

Boolean

```
techGetLayerMaskNumber (
    techFileId      techfile,
    techLayerNum    layerNum,
    techPurpose     purpose,
    techPropType    *pType,
    techPropValue   *pValue
)
```

Description

Gets the mask number assigned to the specified layer.

Note: May accept an LP specification for the first argument, however the maskNumber information is only applicable to the layer. If an LP is specified, the corresponding layer only will be affected. Dynamically changing mask numbers is a dangerous operation that has far reaching implications throughout the technology database and related design databases..

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>*pType</i>	Pointer to the type of the property.
<i>*pValue</i>	Pointer to the mask number applied to the layer.

Return Values

Boolean	Returns TRUE if the mask number assigned to the specified layer is successfully retrieved. Returns FALSE otherwise.

techSetLayerMaskNumber

Boolean

```
techSetLayerMaskNumber (
    techFileId      techfile,
    techLayerNum    layerNum,
    techPurpose     purpose,
    techPropType    pType,
    techPropValue   pValue
)
```

Description

Sets the mask number assigned to the specified layer.

Note: May accept an LP specification for the first argument, however the maskNumber information is only applicable to the layer. If an LP is specified, the corresponding layer only will be affected. Dynamically changing mask numbers is a dangerous operation that has far reaching implications throughout the technology database and related design databases..

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>type</i>	The type of the property.
<i>value</i>	The mask number applied to the layer.

Return Values

Boolean	Returns TRUE if the mask number assigned to the specified layer is successfully set. Returns FALSE otherwise.

techStartGenLayerRoutingGrid

```
techGenStateId  
techStartGenLayerRoutingGrid(  
    techFileId    techfile  
)
```

Description

Starts a generator for examining the routing grids for a given technology database.

Arguments

<i>techfile</i>	The database id of the techfile.

Return Values

<code>techGenStateId</code>	The generator <code>techGenStateId</code> which is used to keep track of the state of the generation of the layer routing grids.

techGenLayerRoutingGrid

Boolean

```
techGenLayerRoutingGrid(  
    techGenStateId    stateId,  
    techLayerNum      *pLayerNum,  
    techPurpose        *pPurpose,  
    techPropType       *pType  
    techPropValue      *pValue,  
    techPropType       *pType2  
    techPropValue      *pValue2,  
    techPropType       *pType3  
    techPropValue      *pValue3  
)
```

Description

With each subsequent call to this function, when the returned value is TRUE, the set of pointer is set to represent the routing grid information stored in the technology database.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<i>stateId</i>	The generator <code>stateId</code> which is used to keep track of the state of the generation of the layer routing grids.
<i>*pLayerNum</i>	The layer number of the layer-purpose pair, returned by indirection.
<i>*pPurpose</i>	The purpose number of the layer-purpose pair. Ignored in OA.
<i>*pType</i>	The layer type, returned by indirection. Refers to the direction data defined for the specified layer in the <code>routingLayers</code> rules of the specified technology file.
<i>*pValue</i>	The layer value, returned by indirection.
<i>*pType2</i>	The layer type, returned by indirection. Refers to the routing pitch of the specified layer in the <code>routingGrids</code> rules of the specified technology file.
<i>*pValue2</i>	The layer value, returned by indirection.
<i>*pType3</i>	The layer type, returned by indirection. Refers to the routing offset defined for the specified layer in the <code>routingGrids</code> rules of the specified technology file.
<i>*pValue3</i>	The layer value, returned by indirection.

Return Values

Boolean	Returns TRUE if a layer routing grid was successfully generated. Returns FALSE otherwise.

techGetLayerRoutingDirection

Boolean

```
techGetLayerRoutingDirection(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *pType,  
    techPropValue   *pValue  
)
```

Description

Gets the routing direction assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>*type</i>	Pointer to the type of the property <i>value</i> .
<i>*value</i>	Pointer to the routing direction applied to the layer.

Return Values

Boolean	Returns TRUE if the routing direction assigned to the specified layer is successfully retrieved. Returns FALSE otherwise.

techSetLayerRoutingDirection

Boolean

```
techSetLayerRoutingDirection(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    type,  
    techPropValue   value  
)
```

Description

Sets the routing direction assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>type</i>	The type of the property.
<i>value</i>	The routing direction applied to the layer.

Return Values

Boolean	Returns TRUE if the routing direction assigned to the specified layer is successfully set. Returns FALSE otherwise.

techGetLayerRoutingPitch

Boolean

```
techGetLayerRoutingPitch(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *pType,  
    techPropValue   *pValue  
)
```

Description

Gets the routing pitch assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>*pType</i>	Pointer to the type of the property.
<i>*pValue</i>	Pointer to the routing pitch applied to the layer.

Return Values

Boolean	Returns TRUE if the routing pitch assigned to the specified layer is successfully retrieved. Returns FALSE otherwise.

techSetLayerRoutingPitch

Boolean

```
techSetLayerRoutingPitch(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    type,  
    techPropValue   value  
)
```

Description

Sets the routing pitch assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>type</i>	The type of the property.
<i>value</i>	The routing pitch applied to the layer.

Return Values

Boolean	Returns TRUE if the routing pitch assigned to the specified layer is successfully set. Returns FALSE otherwise.

techGetLayerRoutingOffset

Boolean

```
techGetLayerRoutingOffset(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *pType,  
    techPropValue   *pValue  
)
```

Description

Gets the routing offset assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>*pType</i>	Pointer to the type of the property.
<i>*pValue</i>	Pointer to the routing offset applied to the layer.

Return Values

Boolean	Returns TRUE if the routing offset assigned to the specified layer is successfully set. Returns FALSE otherwise.

techSetLayerRoutingOffset

Boolean

```
techSetLayerRoutingOffset (
    techFileId      techfile,
    techLayerNum    layerNum,
    techPurpose     purpose,
    techPropType    type,
    techPropValue   value
)
```

Description

Sets the routing offset assigned to the specified layer.

Arguments

<i>techfile</i>	The database identifier of the technology file.
<i>layerNum</i>	The layer number.
<i>purpose</i>	The number of the purpose.
<i>type</i>	The type of the property.
<i>value</i>	The routing offset applied to the layer.

Return Values

Boolean	Returns TRUE if the routing offset assigned to the specified layer is successfully set. Returns FALSE otherwise.

techStartGenLayerFunction

```
techGenStateId  
  
techStartGenLayerFunction(  
    techFileId techfile  
);_
```

Description

Initializes a generator for the specified technology file. A generator state is returned, which is used with the [techGenLayerFunction](#) function.

Arguments

<i>techfile</i>	The database identifier of the technology file.

Return Values

<i>techGenStateId</i>	State ID for the generator.

techGenLayerFunction

Boolean

```
techGenLayerFunction(  
    techGenStateId stateId,  
    techLayerNum *pLayerNum,  
    techPurpose *pPurpose,  
    techPropType *pType,  
    techPropValue *pValue  
) _
```

Description

Retrieves the next technology file ID from the generator with the given techStartGenLayerFunction.

Arguments

<i>stateId</i>	State ID for the generator.
<i>*pLayerNum</i>	The pointer to the layer number.
<i>*pPurpose</i>	The pointer to the purpose.
<i>*pType</i>	Pointer to the type of property. Returned by the function.
<i>*pValue</i>	Pointer to the value of the property. Returned by the function.

Return Value

<i>Boolean</i>	TRUE if the next technology file ID is successfully generated. FALSE if the list is exhausted or an error occurs.

techGenLayerFunctionAndMaskNumber-

Boolean

```
techGenLayerFunctionAndMaskNumber(  
    techGenStateId stateId,  
    techLayerNum *pLayerNum,  
    techPurpose *pPurpose,  
    techPropType *pType,  
    techPropValue *pValue,  
    techPropType *pType2,  
    techPropValue *pValue2  
);
```

Description

Retrieves the next technology file ID and mask number from the generator.

Arguments

<i>stateId</i>	State ID for the generator.
<i>*pLayerNum</i>	Pointer to the layer number.
<i>*pPurpose</i>	Pointer to purpose.
<i>*pType</i>	Pointer to the type of property. Returned by the function.
<i>*pValue</i>	Pointer to the value of property. Returned by the function.
<i>*pType2</i>	Pointer to the type of property. Returned by the function.
<i>*pValue2</i>	Pointer to the value of property. Returned by the function.

Return Value

<i>Boolean</i>	TRUE if the next technology file ID and mask number is successfully generated. FALSE if the list is exhausted or an error occurs.

Layer-Purpose (LP) Pair Functions

A layer-purpose pair is the purpose assigned to a given layer. Separately, a layer and a purpose exist as concepts, not physical entities. A layer and a purpose combined, however, make a physical entity called a “layer-purpose pair.” A layer number and a purpose number define a layer-purpose pair.

An example of a layer-purpose pair is

```
(metall drawing)
```

where `metall` is the layer and `drawing` is the purpose.

A two-layer layer-purpose pair property is a property that is associated with two layers. In the following example,

```
(minSpacing .35 (met1 drawing) (met2 drawing))
```

- `minSpacing` is the property name
- `.35` is the value of the property
- `(met1 drawing)` is the layer-purpose pair for the first layer and `(met2 drawing)` is the layer-purpose pair for the second layer.

Shapes such as polygons, paths, and rectangles can only be created on a layer-purpose pair. Items listed in the LSW (layer selection window) are layer-purpose pairs. You can only select or view an object by its layer-purpose pair.

All layer-purpose pairs that reference the same layer, such as `((metall drawing) (metall pin))` have the same attributes. You create two separate layer-purpose pairs for viewing control. For example, if you only want to see the pin shapes which have been created on the `(metall pin)` layer-purpose pair, with separate layer-purpose pairs created, you can turn all of the other layer-purpose pairs off.

The following technology file functions pertain to layer-purpose pairs.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techContainsLP

Boolean

```
techContainsLP(  
    techFileId    techfile,  
    techLayerNum  layer,  
    techPurpose   purpose  
);
```

Description

Determines whether the technology file contains a layer-purpose pair represented by a layer number and a purpose.

Arguments

techfile	The database identifier of the technology file.
layer	The layer number.
purpose	The specified purpose number defined in the technology file.

Return Values

Boolean	Returns TRUE if the technology file contains the layer-purpose pair described by the layer and purpose. Returns FALSE if the layer-purpose pair does not exist in the technology file.

techCreateDerivedPurposeDef

```
techPurposeDefId  
techCreateDerivedPurposeDef(  
    techPurposeDefId    parentPurposeId,  
    techPurpose          purpose,  
    cppConstString      name,  
    cppConstString      abbrev  
);
```

Description

Creates a purpose definition with the given attributes in the same technology database as the specified parent purpose. The returned purpose will have the parent purpose attribute set.

Arguments

parentPurposeId	The database identifier of the parent purpose.
purpose	The purpose number of the purpose definition.
name	The name of the purpose definition.
abbrev	The abbreviation name of the purpose definition.

Return Values

techPurposeDefId	Returns the database identifier of the newly created purpose definition. Returns <code>techcNullPurposeDefId</code> if the technology file or purpose definition does not exist.

Example

```
techPurposeDefId purposeDef3 =  
techCreateDerivedPurposeDef(drawingPurposeDefId, 3, "userPurposeDef3", "up3");
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techCreateLP

```
techLPId
techCreateLP(
    techFileId      techfile,
    techLayerNum     number,
    techPurpose      purpose,
    String           name
);
```

Description

Creates a layer-purpose pair in the specified technology file. The layer-purpose pair is defined with defaults in the `techLayerPurposePriorities` and `techDisplays` rules in the `layerDefinitions` section of the technology file.

The `techLayerPurposePriorities` rules of the technology file lists layer-purpose pairs in priority order. Layer-purpose pairs listed first are displayed first. Layer-purpose pairs listed later are displayed on top of, and may obscure, layer-purpose pairs already displayed.

The `techDisplays` rules of the technology file define the display attributes of a layer. Layer display attributes include the display packet used, whether the layer is visible or selectable, and if the layer is visible when dragged.

This function also creates a database object for the layer-purpose pair and returns the database identifier. You use the identifier with the `techSetLP` functions and `techSetLPPacket` functions to update the defaults and to specify how the layer-purpose pair appears in your designs. The layer and purpose you specify must be defined in the technology file before you can create a layer-purpose pair.

Default Values

The defaults set by this function are as follows:

Technology File section	Argument	Default
<code>techLayerPurposePriorities</code>	Priority	0
<code>techDisplays</code>	Packet	defaultPacket
	Visibility	TRUE
	Selectability	TRUE

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Technology File section	Argument	Default
	Contributes to changed layer	TRUE
	Drag enable	TRUE
	Valid	TRUE

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>number</code>	The number of the layer in the layer-purpose pair specified in the technology file.
<code>purpose</code>	The purpose name in the layer-purpose pair specified in the technology file.
<code>name</code>	The name of the layer-purpose pair created.

Return Values

<code>techLPId</code>	Returns the database (db) identifier of the newly created layer-purpose pair. Returns <code>techcNullLpId</code> if the technology file, layer, or purpose does not exist or if the layer-purpose pair already exists.

techCreatePurposeDef

```
techPurposeDefId  
techCreatePurposeDef(  
    techFileId      techfile,  
    techPurpose     purpose,  
    cppConstString  name,  
    cppConstString  abbrev  
);
```

Description

This function creates a database object for the purpose definition and returns the database identifier.

You can use the identifier with other functions such as `techSetPurposeDefVoltageRange` to update the attributes of the purpose definition.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>purpose</code>	The purpose number of purpose definition.
<code>name</code>	The name of the purpose definition.
<code>abbrev</code>	The abbreviation name of the purpose definition.

Return Values

<code>techPurposeDefId</code>	Returns the database identifier of the newly created purpose definition. Returns <code>techcNullPurposeDefId</code> if the technology file or purpose definition does not exist.

Example

```
techPurposeDefId purposeDef11 = techCreatePurposeDef(techLib, 11,  
"userPurposeDef11", "p11");
```

techCreateTwoLayerProp

```
techGroupId  
techCreateTwoLayerProp(  
    techLPId      lpId1,  
    techLPId      lpId2,  
    String        propName,  
    techPropType   propType,  
    techPropValue  propValue,  
    Boolean        ordered  
);
```

Description

Creates a two-layer layer-purpose pair property. The property is associated with both layer 1 and layer 2.

Arguments

lpId1	The first layer-purpose pair (LP1).
lpId2	The second layer-purpose pair (LP2).
propName	The property name for the layer-purpose pair.
propType	The property type for the layer-purpose pair.
propValue	The property value for the layer-purpose pair.
ordered	If the layer-purpose pairs are ordered, the value of lp1, lp2 can be different than lp2, lp1.

Return Values

techGroupId	Returns TRUE if the property is ordered, or FALSE if it is not. Returns techcNullGroupId if an error exists.

techCreateTwoLayerPropByNum

```
techGroupId  
techCreateTwoLayerPropByNum(  
    techFileId      techfile,  
    techLayerNum    layer1,  
    techLayerNum    layer2,  
    String          propName,  
    techPropType    propType,  
    techPropValue    propValue,  
    Boolean         ordered  
);
```

Description

Creates a two-layer property, based on the numbers of the layers given.

Arguments

techfile	The database identifier of the technology file.
layer1	The layer number of the first layer.
layer2	The layer number of the second layer.
propName	The property name.
propType	The property type.
propValue	The property value.
ordered	The function returns TRUE if the property is ordered, and FALSE if it is not. If the property is ordered, layer1, layer2 is different from layer2, layer1. If the property is unordered, layer1, layer2 is equivalent to layer2, layer1.

Return Values

techGroupId	Returns techGroupId if the two-layer property is successfully created. Returns techcNullGroupId if an error occurred and the property was not created.

techDeleteLP

Boolean

```
techDeleteLP(  
    techLPId lpId  
);
```

Description

Deletes the specified layer-purpose pair from the technology file associated with the layer-purpose pair. The layer-purpose pair is defined in the `layerDefinitions` section of the technology file. If a database object for the layer-purpose pair does not exist, this function returns `FALSE`.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.

Return Values

<code>Boolean</code>	Returns <code>TRUE</code> if a database object for the layer-purpose pair exists and the layer-purpose pair is successfully deleted. Returns <code>FALSE</code> if a database object for the layer-purpose pair does not exist.

techDeletePurposeDef

Boolean

```
techDeletePurposeDef(  
    techPurposeDefId  purposeDefId  
);
```

Description

Deletes the specified purpose definition from the associated technology file.

Arguments

<code>purposeDefId</code>	The database identifier for the specified purpose definition.

Return Values

Boolean	Returns TRUE if a database object for the purpose definition exists and the purpose definition is successfully deleted. Returns FALSE if a database object for the purpose definition does not exist.

Example

```
techDeletePurposeDef(purposeDef11);
```

techDeleteTwoLayerProp

Boolean

```
techDeleteTwoLayerProp(  
    techLPId    lpId1,  
    techLPId    lpId2,  
    String      propName  
);
```

Description

Deletes the specified two-layer property from the `techLayerProperties` rules of the specified technology file. It specifies special properties that you want to place on the layers in your design.

The `techLayerProperties` rules of the technology file contained user-defined properties for specific layer-purpose objects. The `techLayerProperties` rules are located in `layerDefinitions` section of the technology file.

Arguments

<code>lpId1</code>	The database identifier of the property for the first layer.
<code>lpId2</code>	The database identifier of the property for the second layer.
<code>propName</code>	The name of the two-layer property.

Return Values

Boolean	Returns TRUE if the property is successfully deleted. Returns FALSE if an error occurs or if the technology file or property does not exist.

techDeleteTwoLayerPropByNum

Boolean

```
techDeleteTwoLayerPropByNum(  
    techFileId    techfile,  
    techLayerNum  layer1,  
    techLayerNum  layer2,  
    String        propName  
);
```

Description

Deletes the specified two-layer property.

Arguments

techfile	The database identifier of the technology file.
layer1	The first layer (Layer 1).
layer2	The second layer (Layer 2).
propName	The property name.

Return Values

Boolean	Returns TRUE if the property is successfully deleted. Returns FALSE if an error occurs or if the property doesn't exist.

techGenLP

Boolean

```
techGenLP(  
    techGenStateId  pState,  
    techLayerNum    *pLayer,  
    techPurpose     *pPurpose  
);
```

Description

Retrieves the next layer number and purpose that represent a specified layer-purpose pair. Use this function after [“Virtuoso Layout Suite XL Functions”](#) on page 1369.

Arguments

pState	The generator state identifier.
*pLayer	The pointer to the layer number (LayerNum).
*pPurpose	The pointer to the purpose (purpose).

Return Values

Boolean	Returns TRUE if the values are successfully retrieved.
	Returns FALSE when the function finishes retrieving values.

techGenLPId

Boolean

```
techGenLPId(  
    techGenStateId  pState,  
    techLPId        *lpId  
);
```

Description

Generates the next layer purpose pair.

Arguments

pState	The generator state identifier.
*lpId	A pointer to the layer purpose pair identifier.

Return Values

Boolean	Returns TRUE if the layer purpose pair is successfully generated. Returns FALSE when the generator is finished.

techGenPurposeDef

Boolean

```
techGenPurposeDef(  
    techGenStateId    pState,  
    techPurposeDefId  *purposeDefId  
);
```

Description

Generates the next purpose definition.

Arguments

<code>pState</code>	The generator state identifier <code>techGenStateId</code> , used to track the state of purpose definition generation.
<code>*purposeDefId</code>	A pointer to the purpose definition identifier.

Return Values

Boolean	Returns TRUE if the purpose definition is successfully generated.

Example

```
techGenStateId    stateId = techStartGenPurposeDef(techDB);  
techPurposeDefId  purposeDefId;  
while techGenPurposeDef(stateId, &purposeDefId) {  
}
```

techGetLayerProp

```
techPropId  
techGetLayerProp(  
    techFileId      techfile,  
    cppConstString  layerNum,  
    cppConstString  propName,  
    techPropType    *type,  
    techPropValue   *value  
);
```

Description

Gets the value of the specified layer property in the `techLayerProperties` rules in the technology file.

The `techLayerProperties` rules are located in the `layerDefinitions` section of the technology file. The `techLayerProperties` rules contain user-defined properties for specific layer-purpose objects. It specifies special properties that you want to place on the layers in your design.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerName</code>	The specified layer name.
<code>propName</code>	The specified layer property, taken from the technology file or from the specified data or created by the function.
<code>*type</code>	Pointer to the property type.
<code>*value</code>	Pointer to the property value to be updated.

Return Values

<code>techPropId</code>	Returns the identifier of the layer property. Returns <code>dbcNullPropId</code> if the technology file or layer does not exist.

techGetLayerPropByNumber

```
techPropId  
techSetLayerPropByNumber(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    cppConstString  name,  
    techPropType    *type,  
    techPropValue   *value  
);
```

Description

Gets the property of the layer based on the layer number.

Arguments

techfile	The database identifier of the technology file.
layerNum	The layer number of the layer property.
name	The name of the property.
type	Pointer to the property type.
value	Pointer to the property value.

Return Values

techPropId	Returns the identifier of the property, if the property exists.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techGetLP

```
techLPId  
techGetLP(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Returns the database identifier of the specified layer-purpose pair. If the layer number, purpose, or technology file does not exist, this function returns `techcNullLpId`.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The number of the layer in question. If you have only the <code>layerName</code> argument and need the <code>techLayerNum</code> (the layer number) argument, you can use the <code>techGetLayerNum</code> function. See a description of “techGetLayerNum” on page 1252.
<code>purpose</code>	The specified purpose number defined in the technology file.

Return Values

<code>techLPId</code>	Returns the identifier of the layer-purpose pair if the layer-purpose pair exists. Returns <code>techcNullLpId</code> if the layer-purpose pair does not exist or if there is an error.

techGetLPChangeLayer

Boolean

```
techGetLPChangeLayer(  
    techLPId lpId  
);
```

Description

Determines whether the layer-purpose pair contributes to a changed layer or not. For any editing operations, if the layer-purpose pair of an object contributes to a changed layer, this function adds the old and new value of the layer-purpose pair of an object to the changed layer.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

<code>Boolean</code>	Returns TRUE if the layer-purpose pair contributes to a changed layer. Returns FALSE if the layer-purpose pair does not contribute to a changed layer, if the layer-purpose pair identifier (<code>lpId</code>) is illegal, or if there is an error.

techGetLPDragEnable

Boolean

```
techGetLPDragEnable(  
    techLPId lpId  
);
```

Description

Returns the drag enable attribute that is set on the layer-purpose pair, indicating whether the layer-purpose pair will be dragged or not, during interactive editing commands.

Arguments

lpId	The identifier of the layer-purpose pair.

Return Values

Boolean	Returns TRUE if the function completes successfully. The function is successful if the layer-purpose pair can and will be dragged during interactive editing operations. Returns FALSE if the layer-purpose pair will not be dragged, if lpId is illegal, or if there is an error.

techGetLPPlayerName

```
String  
techGetLPPlayerName(  
    techLPId lpId  
);
```

Description

Retrieves the layer name of a specified layer-purpose pair.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.

Return Values

<code>String</code>	Returns the name of the layer associated with the supplied layer-purpose pair identifier (<code>lpId</code>). Returns <code>dbcNullString</code> if there is an error.

techGetLPLayerProp

Boolean

```
techGetLPLayerProp(  
    techLPId      lpId,  
    String        name,  
    techPropType  *type,  
    techPropValue *value  
);
```

Description

Retrieves the specified property from the specified layer-purpose pair.

Arguments

lpId	The database identifier for the specified layer-purpose pair.
name	The name of the property to retrieve.
*type	The property type, returned by indirection.
*value	The property value, returned by indirection.

Return Values

techGetLPName

```
String  
techGetLPName(  
    techLPId lpId  
);
```

Description

Retrieves the layer name of a specified layer-purpose pair.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.

Return Values

<code>String</code>	Returns the name of the layer associated with the supplied layer-purpose pair identifier (<code>lpId</code>). Returns <code>dbcNullString</code> if there is an error.

techGetLPNumById

```
techLayerNum  
techGetLPNumById(  
    techLPId lpId  
);
```

Description

Retrieves the layer number of a specified layer-purpose pair.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.

Return Values

<code>techLayerNum</code>	Returns the layer number within the range of 0 through 255. Returns <code>techcNullLayer</code> if there is an error.

techGetLPPacketName

```
String  
techGetLPPacketName (  
    techLPId lpId  
);
```

Description

Returns the packet name assigned to the specified layer-purpose pair. The packet name attribute is defined in the `techDisplays` rules of the technology file associated with the layer-purpose pair.

The `techDisplays` rules of the technology file define the display attributes of a layer. Layer display attributes include the display packet, whether the layer is visible or selectable, and if the layer is visible when dragged. These rules must be contained within the `layerDefinitions` section of the technology file.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

<code>String</code>	Returns the name of the packet defined for the layer-purpose pair.
	Returns FALSE if the layer-purpose pair does not exist.

techGetLPPurposeDef

```
techPurposeDefId  
techGetLPPurposeDef (  
    techLPId    lpId  
);
```

Description

Retrieves the purpose definition of a specified layer-purpose pair.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.

Return Values

<code>techPurposeDefId</code>	Returns <code>techcUnknownPurposeDefId</code> if the function is not successful.

Example

```
techPurposeDefId drawingPurposeDef = techGetLPPurposeDef(lpWireDrawing);
```

techGetLPPriority

```
int  
techGetLPPriority (  
    techLPId lpId  
);
```

Description

Returns the drawing priority of the layer-purpose pair. The topmost layer-purpose pair (the layer-purpose pair with the highest number) is drawn on top. Each window, when drawn, references one particular technology file.

You might have five windows, for example. Each window contains a different cellview, each of which reference a different technology file. In this case, you would have five sets of “layer-purpose pair priorities” that are completely independent of each other. Each window would have priorities ranging from 0 to however many layer-purpose pairs exist in the corresponding technology file.

Arguments

lpId	The identifier of the layer-purpose pair.

Return Values

int	The drawing priority of the layer-purpose pair. The values range from zero to the number of layer-purpose pairs in the technology file.

techGetLPPriorityInContext

```
long  
techGetLPPriorityInContext (  
    techFileId    techId,  
    techLPId      lpId  
);
```

Description

Returns the drawing priority of a layer-purpose pair in the effective technology database, down through the graph starting with a specified local technology database.

The topmost layer-purpose pair (the layer-purpose pair with the highest number) is drawn on top. Each window, when drawn, references one particular incremental technology database graph.

You might have five windows, for example. Each window contains a different cellview, each of which reference a different incremental technology database graph. In this case, you would have five sets of “layer-purpose pair priorities” that are completely independent of each other. Each window would have priorities ranging from 0 to however many layer-purpose pairs exist in the corresponding effective technology database.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

<code>long</code>	The drawing priority of the layer-purpose pair in the specified technology database.

techGetLPProp

```
techPropId  
techGetLPProp(  
    techLPId      lpId,  
    String        name,  
    techPropType  *type,  
    techPropValue *value  
);
```

Description

Retrieves the named property from the layer-purpose pair identifier (`lpId`). The values are returned by indirection.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.
<code>name</code>	The name of the property.
<code>*type</code>	The property type, returned by indirection.
<code>*value</code>	The property value, returned by indirection.

Return Values

<code>techPropId</code>	Returns the identifier of the property, if the property exists. Returns <code>dbcNullPropId</code> if the property does not exist.

techGetLPPurpose

```
techPurpose  
techGetLPPurpose(  
    techLPId lpId  
);
```

Description

Retrieves the purpose of a specified layer-purpose pair.

Arguments

<code>lpId</code>	The database identifier for the specified layer-purpose pair.

Return Values

<code>techPurpose</code>	Returns the purpose of the layer-purpose pair. Returns <code>techUnknownPurpose</code> if the function is not successful.

techGetLPSelectable

Boolean

```
techGetLPSelectable (  
    techLPId lpId  
);
```

Description

Returns the selectability setting of the specified layer-purpose pair. If the attribute of the `lpId` (layer-purpose pair identifier) is set to TRUE, the identifier is selectable, if it is set to FALSE, the identifier is not selectable. Objects must have a selectable layer-purpose pair to be selectable themselves.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

Boolean	Returns TRUE if the layer-purpose pair identifier is selectable. Returns FALSE if the layer-purpose pair identifier is not selectable. Not selectable means the objects on this layer-purpose pair cannot be selected and modified during interactive editing operations.

techGetLPTechFile

```
techFileId  
techGetLPTechFile(  
    techLPId lpId  
);
```

Description

Returns the identifier of the technology file (`techFileId`) containing the passed `lpId` (layer-purpose pair identifier).

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

<code>techFileId</code>	Returns the identifier of the technology file containing the layer-purpose pair.

Example

```
techGetLPTechFile();  
techFileId techFile;  
techLPId lpId;  
techFile = techGetLPTechFile (lpId);
```

techGetLPValid

Boolean

```
techGetLPValid(  
    techLPId lpId  
);
```

Description

Returns the valid setting of the layer-purpose pair that was set by the `techSetLPValid` function. Unless overridden, the layer-purpose pair shows up in the LSW (layer selection window).

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

Boolean	Returns TRUE if the layer-purpose pair is valid. Returns FALSE if the <code>lpId</code> is illegal, if the layer-purpose pair is not valid, or if there is an error.

techGetLPVisible

Boolean

```
techGetLPVisible (  
    techLPId lpId  
);
```

Description

Returns the visible setting for a specified layer-purpose pair (set by the `techSetLPVisible` function). A layer-purpose pair must be visible to be drawn or displayed.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.

Return Values

<code>Boolean</code>	Returns TRUE if the layer-purpose pair is visible (all shapes on the layer-purpose pair are drawn). Returns FALSE if the layer-purpose pair is invisible (meaning the shapes on the layer-purpose pair are not drawn), if the layer-purpose pair identifier (<code>lpId</code>) is illegal, or if there is an error.

techGetPurposeAbbrev

String

```
techGetPurposeAbbrev(  
    techFileId  techfile,  
    techPurpose purpose  
);
```

Description

Retrieves the abbreviation of the specified purpose.

Arguments

techfile	The database identifier of the technology file.
purpose	The specified purpose number defined in the technology file.

Return Values

String	Returns the name of the abbreviation. Returns <code>dbcNullString</code> if there is no abbreviation or if a purpose does not exist.

techGetPurposeDef

```
techPurposeDefId  
techGetPurposeDef(  
    techFileId    techfile,  
    techPurpose   purpose  
);
```

Description

Returns the purpose definition in the specified techfile with the specified `techPurpose` number. If the purpose definition doesn't exist, `techcNullPurposeDefId` is returned.

Arguments

<code>techfile</code>	Database identifier of the technology file.
<code>purpose</code>	The specified purpose number defined in the technology file.

Return Values

<code>techPurposeDefId</code>	Returns the database identifier (<code>techPurposeDefId</code>) of the purpose definition. Returns <code>techcNullPurposeDefId</code> if the function fails.

Example

```
techCreatePurposeDef(tf, 11, "userPurposeDef11", "p11");  
techGetPurposeDef(tf, 11);
```

techGetPurposeDefAbbrev

```
String  
techGetPurposeDefAbbrev(  
    techPurposeDefId  purposeDefId  
);
```

Description

Retrieves the abbreviation name of the specified purpose definition.

Arguments

<code>purposeDefId</code>	Database identifier of the purpose definition.

Return Values

<code>String</code>	Returns the name of the abbreviation. Returns <code>techcNullString</code> if there is no abbreviation or if a purpose definition does not exist.

Example

```
String abbrev = techGetPurposeDefAbbrev(userPurposeDefId2);
```


techGetPurposeDefDescription

Boolean

```
techGetPurposeDefDescription(  
    techPurposeDefId    purposeDefId,  
    String              *descriptionString  
);
```

Description

Returns the description attribute value on the specified purpose definition. if the description attribute value is not set, it returns `FALSE`.

Note: It is recommended to verify that the `techPurposeDefHasDescription()` returns `TRUE` before calling this function. This prevents a warning message from appearing if the description attribute is not set.

Note:

Arguments

<code>purposeDefId</code>	Database identifier of the purpose definition.
<code>*descriptionString</code>	Pointer to the specified description string.

Return Values

<code>Boolean</code>	Returns <code>TRUE</code> if successful.
	Returns <code>FALSE</code> if the purpose definition does not exist or does not have the description attribute set.

Example

```
String description;  
techGetPurposeDefDescription(techcNullPurposeDefId, &description);
```

techGetPurposeDefName

```
String  
techGetPurposeDefName(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns the name of the specified purpose definition defined in the `techPurposes` rules of the specified technology file.

The `techPurposes` rules are located in the `layerDefinitions` section of the technology file. The `techPurposes` rules define the purpose that can be matched with a layer to create layer-purpose pairs.)

Arguments

<code>purposeDefId</code>	Database identifier of the purpose definition.

Return Values

<code>String</code>	Returns the purpose name of the specified purpose definition.
	Returns <code>techcNullString</code> if the purpose definition does not exist.

Example

```
String defName = techGetPurposeDefName(userPurposeDefId2);
```

techGetPurposeDefNumber

```
techPurpose  
techGetPurposeDefNumber(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns the purpose number of the specified purpose definition.

Arguments

<code>purposeDefId</code>	Database identifier of the purpose definition.

Return Values

<code>techPurpose</code>	Returns the purpose number of the specified purpose definition.

Example

```
techPurpose purposeNum = techGetPurposeDefNumber(userPurposeDefId2);
```

techGetPurposeDefParent

```
techPurposeDefId  
techGetPurposeDefParent (  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns the parent purpose definition of the specified purpose definition. If the parent purpose attribute of the purpose definition is not set, the purpose definition for the "drawing" purpose is returned.

Arguments

<code>purposeDefId</code>	Database identifier of the purpose definition.

Return Values

<code>techPurposeDefId</code>	Returns the database identifier of the parent purpose definition.

Example

```
techPurposeDefId parentDefId = techGetPurposeDefParent (userPurposeDefId2);
```

techGetPurposeDefSigType

Boolean

```
techGetPurposeDefSigType(  
    techPurposeDefId    purposeDefId,  
    String               *sigTypeString  
);
```

Description

Returns the signal type attribute value of the specified purpose definition. If the signal type attribute value is not set, FALSE is returned.

Note: It is recommended that you verify that the `techPurposeDefHasSigType()` returns TRUE before calling this function. This prevents a warning message from appearing if the signal type attribute is not set.

Arguments

<code>purposeDefId</code>	The database identifier of the specified purpose definition.
<code>sigTypeString</code>	The signal type attribute value of the specified purpose definition.

Return Values

Boolean	Returns TRUE if successful.

Example

```
String  sigTypeName;  
techGetPurposeDefSigType(userPurposeDef3, &sigTypeName);
```

techGetPurposeDefVoltageRange

Boolean

```
techGetPurposeDefVoltageRange(  
    techPurposeDefId    purposeDefId,  
    float                *minVoltage,  
    float                *maxVoltage  
);
```

Description

Returns the voltage range attribute values of the specified purpose definition. If the voltage range attribute value is not set, FALSE is returned.

Note: It is recommended to verify that the `techPurposeDefHasVoltageRange()` returns TRUE before calling this function. This prevents a warning message from appearing if the voltage range attribute is not set.

Arguments

purposeDefId	The database identifier of the specified purpose definition.
*minVoltage	The minimum voltage of the specified purpose definition.
*maxVoltage	The maximum voltage of the specified purpose definition.

Return Values

Boolean	Returns TRUE, if successful.

Example

```
float minVoltage;  
float maxVoltage;  
techGetPurposeDefVoltageRange(userPurposeDef3, &minVoltage, &maxVoltage);
```

techGetPurposeName

String

```
techGetPurposeName(  
    techFileId  techfile,  
    techPurpose purpose  
);
```

Description

Returns the purpose name of the specified purpose number defined in the `techPurposes` rules of the specified technology file.

The `techPurposes` rules are located in the `layerDefinitions` section of the technology file. The `techPurposes` rules define the purposes that can be matched with a layer to create layer-purpose pairs.)

Some applications truncate purpose names when they display layer purpose pairs in lists or forms. If the purpose name is not unique when truncated to three characters, specify an acceptable abbreviation for the applications to use instead.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>purpose</code>	The specified purpose number defined in the technology file.

Return Values

<code>String</code>	Returns the purpose name of the specified purpose number. Returns <code>dbcNullString</code> if the purpose does not exist.

techFindTwoLayerProp

```
techPropId
techFindTwoLayerProp(
    techLPId    lpId1,
    techLPId    lpId2,
    String      propName,
    techPropType *pType,
    techPropValue *pValue,
    Boolean     *pOrdered
);
```

Description

Retrieves a two-layer property and its order, using layer-purpose pairs as arguments.

Arguments

lpId1	The database identifier for the first layer-purpose pair.
lpId2	The database identifier for the second layer-purpose pair.
propName	The name of the two-layer property.
*pType	The property type, returned by indirection.
*pValue	The property value, returned by indirection.
*pOrdered	The pointer to the address of the <code>Boolean</code> <code>ordered</code> argument.

Return Values

techPropId	Returns the database property identifier (<code>techPropId</code>) property.
	Returns <code>dbcNullPropId</code> if the function fails.

Example

```
techFileId techfile;
techLPId lpId1, lpId2;
String propName = "spacing";
techPropType type;
techPropValue value;
techPropId propId;
Boolean ordered;
```


Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

```
lpId1 = techGetLP(techfile, techGetLayerNum(techfile, "met1"),
    techcDrawingPurpose);
lpId2 = techGetLP(techfile, techGetLayerNum(techfile, "met2"),
    techcDrawingPurpose);
propId = techFindTwoLayerProp(lpId1, lpId2, propName, &type, &value, &ordered);
/* Did we find a a prop called "spacing", associated with lp1 and lp2? */
if (propId) {
    switch (type) {
        case dbcIntType:
            /*
             value.aInt = spacing prop value and it's of type integer;
            */
            break;

        case dbcFloatType:
            /*
             value.aFloat = spacing prop value and it's of type float;
            */
            break;

        case ilExprType:
            /*
             it's an expression and we need to ilEval() it. */
            break;

        default:
            /*
             probably not something we were expecting.
            */
            return;
    }

    if (ordered) {
        /*
         passed in order lp1, lp2 makes a difference. Had we passed in
         lp2, lp1, would have got something else or nothing at all.
        */
    }
    else {
        /*
         order doesn't matter. passing in lp1, lp2 or lp2, lp1 returns
         the same propId.
        */
    }

    /* do important stuff with the prop and value. */

    return;
}
```

techFindTwoLayerPropByNum

```
techPropId
techFindTwoLayerPropByNum(
    techFileId      techfile,
    techLayerNum    layer1,
    techLayerNum    layer2,
    String          propName,
    techPropType    *pType,
    techPropValue   *pValue,
    Boolean         *pOrdered
);
```

Description

Retrieves a two-layer property and its order, using layer numbers as arguments.

Arguments

techfile	The database identifier of the technology file.
layer1	The first layer.
layer2	The second layer.
propName	The property name to find.
*pType	The property type, returned by indirection.
*pValue	The property value, returned by indirection.
*pOrdered	The pointer to the address of the <code>Boolean</code> <code>ordered</code> argument.

Return Values

techPropId	Returns the database property identifier (<code>techPropId</code>) of the property. Returns <code>dbcNullPropId</code> if the function fails.

techIsReservedPurposeDef

Boolean

```
techIsReservedPurposeDef(  
    techPurposeDefId  purposeDefId  
);
```

Description

Returns `TRUE` when the specified purpose definition refers to a Virtuoso reserved purpose number.

The purpose number of a Virtuoso reserved purpose definition, is in the range of OpenAccess reserved purposes(from `0xffff0000` to `0xffffffff`) and the Virtuoso system purposes (from `techcMinPurpose` to `techcMaxPurpose`).

Arguments

<code>purposeDefId</code>	The database identifier of the specified purpose definition.

Return Values

Boolean	Returns <code>TRUE</code> if the specified purpose is a Virtuoso reserved purpose.

Example

```
Boolean isSystemReserved = techIsReservedPurposeDef(techGetPurposeDef(tf,  
"drawing"));
```

techList2Layer

```
techLayerNum
techList2Layer(
    String      funcName,
    list        l,
    techFileId  techfile
);
```

Description

This function retrieves a layer identifier from a technology file from a list argument. (Extracts a layer from a list.)

This function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code.

Arguments

funcName	The name of the calling function. If an error occurs, the function name is used in the error or warning message.
l	The list containing the layer. The layer can be a number (25) or name (<code>metal1</code>).
techfile	The name of the technology file.

Return Values

techLayerNum	Returns the number of the layer supplied in the list argument. Returns <code>techcNullLayer</code> if there is an error.

techListToLayer

```
techLayerNum  
techListToLayer(  
    String      funcName,  
    list        l,  
    techFileId  techfile  
);
```

Description

This function retrieves a layer identifier from a technology file from a list argument. (Extracts a layer from a list.)

The function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code.

This function achieves the same result as `techList2Layer`, but with a different implementation. (This is a slower version of the `techList2Layer` function.)

Arguments

<code>funcName</code>	The name of the calling function. If an error occurs, the function name appears in the error or warning message.
<code>l</code>	The list containing the layer. The layer can be a number (25) or name (<code>metal1</code>).
<code>techfile</code>	The database identifier of the technology file.

Return Values

<code>techLayerNum</code>	Returns the number of the layer supplied in the list argument. Returns <code>techcNullLayer</code> if there is an error.

techList2LP

Boolean

```
techList2LP(  
    String      funcName,  
    list        l,  
    techLayerNum *pLayer,  
    techPurpose *pPurpose,  
    techFileId  techfile  
);
```

Description

Like `techList2TechFile`, this function retrieves the layer-purpose pair identifier from a technology file from a list argument. (Extracts a layer-purpose pair from a list.) The layer-purpose pair must be extracted by indirection because two items, `LayerNum` and `Purpose`, are being extracted.

This function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code. This function is used when getting an argument from SKILL.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

funcName	The function name used. If an error exists, the function name is used in the error or warning message. The name of the calling function. If an error occurs, the function name is used in the error or warning message.
l	The list containing the layer-purpose pair identifier, such as (metal1 "drawing"). The list containing the techFileId.
*pLayer	The pointer to the address of the layer number.
*pLayer	The layer number (returned by indirection). The value is legal only if the function is successful (if it returns TRUE).
*pPurpose	The purpose (returned by indirection). The value is legal only if the function is successful (if it returns TRUE).
*pPurpose	The pointer to the address of the purpose. If only the layer name is supplied in the list, the purpose is techcAllPurpose.
techfile	The database identifier of the technology file.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE if there is an error or if the technology file identifier is invalid.

techListToLP

Boolean

```
techListToLP(  
    String      funcName,  
    list        l,  
    techLayerNum *pLayer,  
    techPurpose  *pPurpose,  
    techFileId   techfile  
);
```

Description

This function retrieves the layer-purpose pair identifier from a technology file from a list argument. (It extracts a layer-purpose pair from a list.) The layer-purpose pair must be extracted by indirection because two items, `LayerNum` and `Purpose`, are being extracted.

This function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code. This function is used when getting an argument from SKILL.

This function achieves the same result as `techList2LP`, but with a different implementation. For one, this is a slower version of the `techList2LP` function. Functions with 2 in the name (such as `techList2LP`) are faster than functions with `To` in the name (such as `techListToLP`).

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

funcName	The function name used. If an error exists, the function name is used in the error or warning message. The name of the calling function. If an error occurs, the function name is used in the error or warning message.
l	The list containing the <code>techFileId</code> .
*pLayer	The layer number (returned by indirection). The value is legal only if the function is successful (if it returns TRUE). The pointer to the address of the layer number.
*pPurpose	The purpose (returned by indirection). The value is legal only if the function is successful (if it returns TRUE). The pointer to the address of the purpose. If only the layer name is supplied in the list, the purpose is <code>techcAllPurpose</code> .
techFileId	The database identifier of the technology file.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE if there is an error or if the technology file identifier is invalid.

techList2LPNoWarn

Boolean

```
techList2LPNoWarn(  
    String      funcName,  
    list        l,  
    techLayerNum *pLayer,  
    techPurpose *pPurpose,  
    techFileId  techfile  
);
```

Description

Like `techList2TechFile` and `techListToTechFile`, this function retrieves the layer-purpose pair identifier from a technology file from a list argument. (Extracts a layer-purpose pair from a list.) The layer-purpose pair must be extracted by indirection because two items, `LayerNum` and `Purpose`, are being extracted.

This function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code. This function is used when getting an argument from SKILL.

This function is the same as `techList2LP`, but if an error occurs, no automatic error or warning message is generated. However, if `FALSE` is returned, you know there is a problem, so you can generate your own, more meaningful error message.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>funcName</code>	The name of the calling function. If an error occurs, the function name is used in the error or warning message. The function name used. If an error exists, the function name is used in the error or warning message.
<code>l</code>	The list containing the <code>techFileId</code> . 1. The list containing the layer-purpose pair.
<code>*pLayer</code>	The layer number (returned by indirection). The value is legal only if the function is successful (if it returns TRUE).The pointer to the address of the layer number.
<code>*pPurpose</code>	The purpose (returned by indirection). The value is legal only if the function is successful (if it returns TRUE). The pointer to the address of the purpose. If only the layer name is supplied in the list, the purpose is <code>techcAllPurpose</code> .
<code>techFileId</code>	The database identifier of the technology file.

Return Values

<code>Boolean</code>	Returns TRUE if successful. Returns FALSE if there is an error or if the technology file identifier is invalid.

techList2TechFile

```
techFileId  
techList2TechFile(  
    String    funcName,  
    list      l  
);
```

Description

Retrieves the database identifier from a technology file from a list argument. This function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code. Extracts the `techFileId` from list. Used when getting an argument from SKILL.

Retrieves the database identifier from a technology file from a list argument. This function is usually used in SKILL wrapper functions, in interfaces between SKILL and C-level code. This function achieves the same result as `techList2TechFile`, but this function is implemented slightly differently. This is a slower version of the `techList2TechFile` function.

Arguments

<code>funcName</code>	The function name used. If an error exists, the function name is used in the error or warning message.
<code>l</code>	The list containing the <code>techFileId</code> .

Return Values

<code>techFileId</code>	Returns the database identifier of the technology file.

techListToTechFile

```
techFileId  
techListToTechFile(  
    String    funcName,  
    list      l  
);
```

Description

Extracts the `techFileId` from a list argument. Used when getting an argument from SKILL. This is a slower version of the `techList2TechFile` function.

Arguments

<code>funcName</code>	The function name used. If an error exists, the function name is used in the error or warning message.
<code>l</code>	The list containing the <code>techFileId</code> .

Return Values

<code>techFileId</code>	Returns the database identifier of the technology file.

techMakeEffectiveLPsLocal

Boolean

```
techMakeEffectiveLPsLocal(  
    techFileId    techId  
);
```

Description

Adds all layer-purpose pairs in the graph that are currently being used for the local technology database to the local technology database.

Arguments

techId	The database identifier of the technology database.

Return Values

Boolean	Returns TRUE if the function is successful. Returns FALSE otherwise.

techPurposeToString

```
String  
techPurposeToString(  
    techFileId  techfile,  
    techPurpose purpose  
);
```

Description

Retrieves the purpose name. This function achieves the same result as [“techGetPurposeName”](#) on page 1319.

Arguments

techfile	The database identifier of the technology file.
purpose	The specified purpose number defined in the technology file. (This is the purpose number you want to get the name of.)

Return Values

String	Returns the name of the purpose. Returns <code>dbcNullString</code> if the purpose does not exist or if the technology file is invalid.

techReplaceTwoLayerProp

```
techGroupId  
techReplaceTwoLayerProp(  
    techLPId      lpId1,  
    techLPId      lpId2,  
    String        propName,  
    techPropType   propType,  
    techPropValue  propValue,  
    Boolean        ordered  
);
```

Description

Creates a new two-layer property or updates an existing one, using layer-purpose pairs as arguments.

Arguments

layer1	The first layer.
layer2	The second layer.
propName	The property name.
propType	The property type of the property value.
propValue	The property value.
ordered	Returns TRUE if the property is ordered, or FALSE if it is not.

Return Values

techGroupId	Returns the identifier of the group holding the two-layer property. Returns <code>techcNullGroupId</code> if the function fails.

techReplaceTwoLayerPropByNum

```
techGroupId  
techReplaceTwoLayerPropByNum(  
    techFileId      techfile,  
    techLayerNum    layer1,  
    techLayerNum    layer2,  
    String          propName,  
    techPropType    propType,  
    techPropValue   propValue,  
    Boolean         ordered  
);
```

Description

Creates a new two-layer property, or updates an existing one.

Arguments

techfile	The database identifier of the technology file.
layer1	The first layer.
layer2	The second layer.
propName	The property name.
propType	The property type of the property value.
propValue	The property value.
ordered	If ordered is set to TRUE, the property is ordered. If ordered is set to FALSE, the property is not ordered. If the property is ordered the order layer1, layer2 is different from layer2, layer1 order.

Return Values

techGroupId	Returns the identifier of the group holding the two-layer property. Returns techcNullGroupId if the function fails.

techSetLayerPropByNumber

```
techPropId  
techSetLayerPropByNumber(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    cppConstString  name,  
    techPropType    type,  
    techPropValue    value  
);
```

Description

Sets the property of the layer based on the layer number.

Arguments

techfile	The database identifier of the technology file.
layerNum	The layer number of the layer property.
name	The name of the property.
type	The property type.
value	The property value.

Return Values

techPropId	Returns the identifier of the property, if the property exists.

techSetLayerProp

```
techPropId  
techSetLayerProp(  
    techFileId      techfile,  
    cppConstString  layerName,  
    cppConstString  propName,  
    techPropType    type,  
    techPropValue    value  
);
```

Description

Updates the value of the specified layer property in the `techLayerProperties` rules in the technology file.

The `techLayerProperties` rules are located in the `layerDefinitions` section of the technology file. The `techLayerProperties` rules contain user-defined properties for specific layer-purpose objects. It specifies special properties that you want to place on the layers in your design.

If the `techLayerProperties` rules do not exist, this function creates a rule with the specified data. If the property does not exist, the function creates a new layer property and sets the value to the specified value.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerName</code>	The specified layer name.
<code>propName</code>	The specified layer property, taken from the technology file or from the specified data or created by the function.
<code>type</code>	The property type.
<code>value</code>	The property value to be updated.

Return Values

<code>techPropId</code>	Returns the identifier of the property, if the property exists. If the technology file or layer does not exist, this function creates a new layer property and sets the specified value.

techSetLPChangeLayer

Boolean

```
techSetLPChangeLayer(  
    techLPId    lpId,  
    Boolean     changeLayer  
);
```

Description

Sets the layer-purpose pair to either contribute or not to contribute to the changed layer. Sets the change-layer attribute of the lpId (layer-purpose pair identifier) to TRUE if the layer-purpose pair “contributes to the changed layer,” or to FALSE if it does not.

Note: During editing, DIVA keeps track of the area or areas changed. If a layer-purpose pair “contributes to the changed layer,” such as if the `changeLayer` attribute is set to TRUE, any shapes created on the layer-purpose pair when edited are tracked in the `changeLayer` attribute. For more information on DIVA, see the *Diva Reference*.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>changeLayer</code>	If the <code>changeLayer</code> attribute of the <code>lpId</code> (layer-purpose pair identifier) is set to TRUE, the layer-purpose pair contributes to the changed layer during editing operations. If it is set to FALSE, the layer-purpose pair cannot contribute.

Return Values

Boolean	Returns TRUE if is successful.
	Returns FALSE if the <code>lpId</code> is illegal or if there is an error.

Example

```
Boolean status;  
techLPId lpId;  
status = techSetLPChangeLayer(lpId, TRUE);
```

techSetLPDragEnable

Boolean

```
techSetLPDragEnable(  
    techLPId    lpId,  
    Boolean     dragEnable  
);
```

Description

Sets the layer-purpose pair to either be dragged or not be dragged during interactive editing commands. Sets the drag enable attribute of the `lpId` (layer-purpose pair identifier) to TRUE if the shapes on the layer-purpose pair are to be dragged, or to FALSE if they are not.

Arguments

1 Return Values

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>dragEnable</code>	If the drag enable attribute is set to TRUE, the shapes on the layer-purpose pair are dragged (highlighted and moved) during editing operations. If the drag enable attribute is set to FALSE, the shapes are not dragged.
<code>Boolean</code>	Returns TRUE if successful. Returns FALSE if the <code>lpId</code> is illegal or if there is an error.

Example

```
Boolean status;  
techLPId lpId;  
status = techSetLPDragEnable(lpId, TRUE);
```

techSetLPLayerProp

```
techPropId  
techSetLPLayerProp(  
    techLPId      lpId,  
    String        propName,  
    techPropType  type,  
    techPropValue value  
);
```

Description

Creates or updates the property type and value of the supplied layer-purpose pair identifier (lpId).

Arguments

lpId	The layer-purpose identifier (lpId) of the property.
propName	The property name.
type	The property type.
value	The property value.

Return Values

techPropId	Returns the identifier of the property, if the property exists.

techSetLPPacketName

Boolean

```
techSetLPPacketName(  
    techLPId    lpId,  
    String      packet  
);
```

Description

Updates the packet name assigned to the specified layer-purpose pair in the `techDisplays` rules of the technology file associated with the layer-purpose pair. The packet must be defined in the display resource file before you can assign it to an layer-purpose pair.

The `techDisplays` rules of the technology file define the display attributes of a layer. Layer display attributes include the display packet used, whether the layer is visible or selectable, and if the layer is visible when dragged. These rules must be contained within the `layerDefinitions` section of the technology file.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>packet</code>	The packet name of the packet defined in the <code>display.drf</code> file to assign to <code>lpId</code> .

Return Values

<code>Boolean</code>	Returns TRUE if the packet name is successfully updated. Returns FALSE if the technology file, packet, or layer-purpose pair does not exist.

Example

```
Boolean status;  
techLPId lpId;  
status = techSetLPPacketName(lpId, "bluezigzag");)
```

techSetLPPriority

Boolean

```
techSetLPPriority(  
    techLPId    lpId,  
    unsigned int priority  
);
```

Description

Sets the drawing priority of the layer-purpose pair. The higher the number, the higher the priority. The highest priority is drawn on top of all other layer-purpose pairs.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>priority</code>	The drawing priority of the layer-purpose pair to be set.

Return Values

Boolean	Returns TRUE if is successful.
	Returns FALSE if the <code>lpId</code> is illegal or if there is an error.

techSetLPPriorityInContext

```
techLPIId  
techSetLPPriorityInContext(  
    techFileId      techId,  
    techLPIId       lpId,  
    unsigned long    priority  
);
```

Description

Sets the drawing priority of a layer-purpose pair in a effective technology database. The higher the number, the higher the priority. The highest priority is drawn on top of all other layer-purpose pairs. The layer-purpose pair must exist in the graph rooted at the specified technology database and may not exist in the local technology database.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>lpId</code>	The identifier of the layer-purpose pair.
<code>priority</code>	The drawing priority to set for the layer-purpose pair.

Return Values

<code>techLPIId</code>	Returns the identifier of the layer-purpose pair.

techSetLPProp

Boolean

```
techSetLPProp(  
    techLPId      lpId,  
    String        name,  
    techPropType  type,  
    techPropValue value  
);
```

Description

Creates or updates a named property of type and value on the specified layer-purpose pair.

Arguments

lpId	The database identifier for the specified layer-purpose pair.
name	The name of the property to create or update.
type	The type of the property to create or update.
value	The value for the property to create or update.

Return Values

Boolean	Returns TRUE if successful. Returns FALSE if there is an error or if the layer-purpose pair identifier (lpId) is illegal.

techSetLPSelectable

Boolean

```
techSetLPSelectable(  
    techLPId    lpId,  
    Boolean     selectability  
);
```

Description

Sets the selectable attribute of the `lpId` (layer-purpose pair identifier) to TRUE if the layer-purpose pair is selectable, or to FALSE if it is not.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>selectability</code>	If the <code>selectable</code> attribute of the <code>lpId</code> (layer-purpose pair identifier) is set to TRUE, the layer-purpose pair can be selected during editing operations. If it is set to FALSE, the layer-purpose pair cannot be selected.

Return Values

Boolean	Returns TRUE if the layer-purpose pair is selectable. Returns FALSE if <code>lpId</code> (the layer-purpose pair identifier) is illegal or if there is an error.

Example

```
Boolean status;  
techLPId lpId;  
status = techSetLPSelectable(lpId, FALSE);
```

techSetLPValid

Boolean

```
techSetLPValid(  
    techLPId    lpId,  
    Boolean     valid  
);
```

Description

Sets the valid setting of the layer-purpose pair. A layer-purpose pair must be valid to be used in creation and editing operations.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>valid</code>	If the valid setting is set to TRUE, the layer is valid. If the valid setting is set to FALSE, the layer is not valid.

Return Values

Boolean	Returns TRUE if the update of the valid setting is successful. Returns FALSE if <code>lpId</code> (the layer-purpose pair identifier) is illegal or if there is an error.

Example

```
Boolean status;  
techLPId lpId;  
status = techSetLPValid (lpId, FALSE);
```

techSetLPVisible

Boolean

```
techSetLPVisible(  
    techLPId    lpId,  
    Boolean     visibility  
);
```

Description

Sets a visible attribute of the layer-purpose pair.

Arguments

<code>lpId</code>	The identifier of the layer-purpose pair.
<code>visibility</code>	If visibility is set to TRUE, the layer-purpose pair is visible (drawn during the display). If visibility is set to FALSE, the layer-purpose pair is invisible (not drawn during display).

Return Values

Boolean	Returns TRUE if it is successful.
	Returns FALSE if <code>lpId</code> (the layer-purpose pair identifier) is illegal or if there is an error.

Example

```
Boolean status;  
techLPId lpId;  
status = techSetLPVisible (lpId, TRUE);
```

techSetPurposeAbbrev

Boolean

```
techSetPurposeAbbrev(  
    techFileId    techfile,  
    techPurpose   purpose,  
    String        abbrev  
);
```

Description

Updates the abbreviation of the specified purpose in the specified technology file.

Some applications truncate purpose names when they display layer-purpose pairs in lists or forms. If the purpose name is not unique when truncated to three characters, specify an acceptable abbreviation for the applications to use instead.

The `techPurposes` rules of the technology file define the purposes that can be used to define a layer-purpose pair. They refer to the purposes that can be matched with a layer to create layer-purpose pairs. You can indicate the purpose to update by specifying the layer name or number. The `techPurposes` rules are located in the `layerDefinitions` section of the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>purpose</code>	The specified purpose number defined in the technology file.
<code>abbrev</code>	The old abbreviation of the purpose.

Return Values

Boolean	Returns TRUE if the abbreviation is updated successfully. Returns FALSE if the technology file does not exist or the purpose is not defined.

techSetPurposeDefAbbrev

Boolean

```
techSetPurposeDefAbbrev(  
    techPurposeDefId    purposeDefId,  
    cppConstString      abbrev  
);
```

Description

Updates the abbreviation of the specified purpose definition in the technology file.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.
<code>abbrev</code>	The abbreviation name of the purpose definition.

Return Values

Boolean	Returns TRUE if the abbreviation is updated successfully. Returns FALSE if the technology file does not exist or the purpose definition is not defined.

Example

```
Boolean abbrevIsSet = techSetPurposeDefAbbrev(userPurposeDef3, "up3");
```

techSetPurposeDefDescription

Boolean

```
techSetPurposeDefDescription(  
    techPurposeDefId    purposeDefId,  
    descriptionString    value  
);
```

Description

Sets the description attribute of the specified purpose definition.

Note: `descriptionString` should not be an empty string. You can use `techUnsetPurposeDefDescription()` to remove an existing description string.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.
<code>value</code>	The value of the description attribute of the purpose definition.

Return Values

Boolean	Returns TRUE if the description is updated successfully.

Example

```
Boolean descriptionIsSet = techSetPurposeDefDescription(userPurposeDef3, "power  
0.3V");
```


techSetPurposeDefName

Boolean

```
techSetPurposeDefName (
    techPurposeDefId    purposeDefId,
    cppConstString      name
);
```

Description

Updates the name of the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.
<code>name</code>	The specified purpose definition name.

Return Values

Boolean	Returns TRUE if the purpose definition name is updated successfully.
	Returns FALSE if the purpose definition is not defined.

Example

```
techSetPurposeDefName (userPurposeDef3, "tmp222");
techSetPurposeDefName (userPurposeDef3, "userPurposeDef3");
```

techSetPurposeDefParent

Boolean

```
techSetPurposeDefParent (
    techPurposeDefId  purposeDefId,
    techPurposeDefId  parentPurpDefId
);
```

Description

Sets the parent purpose definition attribute of the specified purpose definition.

Note: The specified purpose definition must be one of the OpenAccess built-in purposes.

If `sigType` attribute has already been specified for the purpose definition, the specified parent purpose definition needs to satisfy the following criteria:

- If `sigType` is signal or user-defined `sigType`, then any purpose can be a parent purpose definition.
- If `sigType` is an OpenAccess built-in `sigType`, such as power or ground, other than signal, the parent purpose must be drawing purpose.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.
<code>parentPurpDefId</code>	The database identifier of the parent purpose definition.

Return Values

Boolean	Returns TRUE if the parent purpose definition number is set successfully.

Example

```
Boolean parentPurposeDefIsSet = techSetPurposeDefParent (userDefPurpose3,
slotPurposeDef);
```

techSetPurposeDefSigType

Boolean

```
techSetPurposeDefSigType(  
    techPurposeDefId    purposeDefId,  
    sigTypeString       value  
);
```

Description

Sets the signal type attribute of the specified purpose definition.

Note: The `sigTypeString` attribute value should not be an empty string.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.
<code>value</code>	The value of signal type attribute.

Return Values

Boolean	Returns <code>TRUE</code> if the signal type attribute is set successfully.

Example

```
Boolean parentPurposeDefIsSet =  
techSetPurposeDefParent(userDefPurpose3, drawingPurposeDef);  
Boolean sigTypeIsSet = techSetPurposeDefSigType(userDefPurpose3, "power");
```

techSetPurposeDefVoltageRange

Boolean

```
techSetPurposeDefVoltageRange (  
    techPurposeDefId    purposeDefId,  
    float               minVoltage,  
    float               maxVoltage  
);
```

Description

Sets the maximum voltage attribute of the purpose. The value must be greater than the minimum voltage attribute value, if specified. If the minimum voltage value is not set, the maximum voltage value must be a positive value.

Arguments

purposeDefId	The database identifier of the purpose definition.
minVoltage	The minimum voltage of the specified purpose definition.
maxVoltage	The maximum voltage of the specified purpose definition.

Return Values

Boolean	Returns <code>TRUE</code> if the voltage range attribute is set successfully.

Example

```
Boolean voltRangeIsSet = techSetPurposeDefVoltageRange(userDefPurpose3, 0.1, 0.3);
```

techSetPurposeName

Boolean

```
techSetPurposeName(  
    techFileId    techfile,  
    techPurpose   purpose,  
    String        name  
);
```

Description

Updates the name of the specified purpose number in the specified technology file. It lists the purposes that can be matched with a layer to create layer-purpose pairs.

The `techPurposes` rules of the technology file define the purposes that can be used to define a layer purpose object. The `techPurposes` rules are located in `layerDefinitions` section of the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>purpose</code>	The specified purpose number defined in the technology file.
<code>name</code>	The specified purpose name.

Return Values

Boolean	Returns TRUE if the name of the purpose number is updated successfully. Returns FALSE if the technology file does not exist or the purpose number is not defined.

techStartGenLPId

```
techGenStateId  
techStartGenLPId(  
    techFileId  techfile  
);
```

Description

Starts generating the layer-purpose pairs in the specified technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Values

<code>techGenStateId</code>	Returns the generator state identifier.

techStartGenLP

```
techGenStateId  
techStartGenLP(  
    techFileId techfile  
);
```

Description

Starts the generator for iterating through all layer-purpose pairs in a technology file. Use this function in conjunction with [“techGenLP”](#) on page 1288.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Values

<code>techGenStateId</code>	Returns the generation state identifier.

techStartGenPurposeDef

```
techGenStateId  
techStartGenPurposeDef(  
    techFileId    techfileId  
);
```

Description

Starts the generator for iterating through all purpose definitions in a technology file.

Arguments

<code>techfileId</code>	The database identifier of the technology file.

Return Values

<code>techGenStateId</code>	Returns the generation state identifier.

Example

```
techStartGenPurposeDef(tf);
```


techUnsetPurposeDefDescription

```
void  
techUnsetPurposeDefDescription(  
    techPurposeDefId  purposeDefId  
);
```

Description

Unsets the description attribute of the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

<code>void</code>	Returns <code>void</code> , if the description attribute is successfully unset.

Example

```
techSetPurposeDefDescription(userPurposeDef3, "a user-defined purposeDef");  
techUnsetPurposeDefDescription(userPurposeDef3);
```

techUnsetPurposeDefParent

```
void  
techUnsetPurposeDefParent (  
    techPurposeDefId  purposeDefId  
);
```

Description

Unsets the parent attribute of the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

<code>void</code>	Returns <code>void</code> , if the parent attribute is successfully unset.

Example

```
techSetPurposeDefParent (userPurposeDef3, fillOPCPurposeDef);  
techUnsetPurposeDefParent (userPurposeDef3);
```

techUnsetPurposeDefSigType

```
void  
techUnsetPurposeDefSigType (  
    techPurposeDefId  purposeDefId  
);
```

Description

Unsets the signal type attribute of the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

<code>void</code>	Returns <code>void</code> , if the signal type attribute is successfully unset.

Example

```
techSetPurposeDefSigType (userPurposeDef3, "power");  
techUnsetPurposeDefSigType (userPurposeDef3);
```

techUnsetPurposeDefVoltageRange

```
void  
techUnsetPurposeDefVoltageRange (  
    techPurposeDefId  purposeDefId  
);
```

Description

Unsets the maximum voltage attribute of the specified purpose definition.

Arguments

<code>purposeDefId</code>	The database identifier of the purpose definition.

Return Values

<code>void</code>	Returns <code>void</code> , if the voltage attribute is successfully unset.

Example

```
techSetPurposeDefVoltageRange(userPurposeDef3, 0.1, 0.3);  
techUnsetPurposeDefVoltageRange(userPurposeDef3);
```

techVerifyTechFileId

Boolean

```
techVerifyTechFileId(  
    techId  techfile  
);
```

Description

Verifies that the technology file is valid.

Arguments

techfile	The database identifier of the technology file.

Return Values

Boolean	Returns TRUE if the technology file is valid. Returns FALSE if it is not.

Layout Suite L Rule Functions

The following functions affect the Layout Editor rules.

techStartGenLeLswLayer

```
techGenStateId  
techStartGenLeLswLayer(  
    techFileId techfile  
);
```

Description

Returns the `leLswLayers` layers from the technology database identified by *techfile*, if found. If `leLswLayers` is not found in this technology database, the function searches any open technology database in the graph for `leLswLayers` and returns the first set that is found. The function does not open any technology database that is explicitly closed.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used with <code>techGenLeLswlayer</code> .

techGenLeLswLayer

```
techGenLeLswLayer(  
    techGenStateId    stateId,  
    techLayerNum      *pLayerNum,  
    techPurpose        *pPurpose  
);
```

Description

Generates the next `Lsw` layer (which is a layer-purpose pair).

Arguments

<code>stateId</code>	State identifier returned from the <code>StartGenLeLswLayer</code> function.
<code>*pLayerNum</code>	The layer number of the next <code>Lsw</code> layer.
<code>*pPurpose</code>	The purpose number of the next <code>Lsw</code> layer.

Return Value

<code>t</code>	Returns the first <code>Lsw</code> layer.

techSetLeLswLayer

Boolean

```
techSetLeLswLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Appends the specified layer to the end of the `leLswLayers` constraints in the specified technology file if the layer does not already exist. If the layer does exist, no action is taken.

The `leLswLayers` constraints refer to the layer purpose pairs that display in your designs in the order in which they appear in LSW. If the `leLswLayers` constraint does not exist, the function creates it with the specified data.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair to add.
<code>purpose</code>	The purpose number of the layer-purpose pair to add.

Return Value

Boolean	Returns FALSE if the technology file or the layer-purpose pair does not exist.

Virtuoso Layout Suite XL Functions

The following functions belong to the `virtuosoDefaultExtractorSetup` constraint group in the technology file.

techStartGenLxExtractLayer

```
techGenStateId  
techStartGenLxExtractLayer(  
    techFileId techfile  
);
```

Description

Starts the generator for Virtuoso XL (Lx) extract layers.

The function finds the `virtuosoDefaultExtractorSetup` constraint group in the techfile. If found, the function then searches the constraint group and gets the value of the `validLayers` constraint. The `validLayers` constraint is an array of layers. The function then starts the generator for Virtuoso XL (Lx) extract layers.

The generator then searches for the `validVias` constraint and retrieves the value. The `validVias` constraint is an array of `viaDefs`. For each `viaDef` in the array, the generator returns the cut layer as an `lxExtractorLayer`.

`lxExtractorLayers` is an unordered layer set. Duplicates are filtered out during generation. The layer purpose is `techcAllPurpose`.

For information on Virtuoso XL extract layers, see *Virtuoso Layout Suite XL User Guide*.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

`techfile` The database identifier of the technology file.

Return Value

`techGenStateId` Returns the generator state identifier to be used with `techGenLxExtractLayer`.

 Returns a NULL generator state if the `virtuosoDefaultExtractorSetup` constraint group is not found.

techGenLxExtractLayer

Boolean

```
techGenLxExtractLayer(  
    techGenStateId    stateId,  
    techLayerNum      *layerNum,  
    techPurpose       *purpose  
);
```

Description

Generates the Virtuoso XL (Lx) extract layers.

The generator uses the array of layers in the `validLayers` constraint. The generator returns one layer at a time until it exhausts all layers.

Arguments

<code>stateId</code>	The generator state identifier returned from <code>StartGenLxExtractLayer</code> by indirection.
<code>*pLayerNum</code>	The layer number of the next extract layer, returned by indirection.
<code>*pPurpose</code>	The purpose number of the next extract layer, returned by indirection.

Return Value

Boolean	Returns TRUE if it performs successfully, and the values for the layer and purpose are assigned and valid. Returns FALSE when the generator is finished.

techSetLxExtractLayer

Boolean

```
techSetLxExtractLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Adds the layer to the `validLayers` constraint layer array in the `virtuosoDefaultExtractorSetup` constraint group. If either the constraint group or the constraint does not exist, the function creates it and adds the layer to the layer array value of the constraint. Layer purpose is ignored.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair.
<code>purpose</code>	The purpose number of the layer-purpose pair.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

techIsLxExtractLayer

Boolean

```
techIsLxExtractLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Indicates whether the specified layer is in the `validLayers` constraint layer array or is a cut layer in the `validVias` constraint group.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair to query. The layer must be defined in the <code>lxExtractLayers</code> constraints of the technology file.
<code>purpose</code>	The purpose number of the layer-purpose pair to query. The layer must be defined in the <code>lxExtractLayers</code> constraints of the technology file.

Return Value

Boolean	Returns TRUE if the specified layer is in the <code>validLayers</code> constraint layer array or is a cut layer in the <code>validVias</code> constraint group. Returns FALSE otherwise.

techStartGenLxNoOverlapLayer

```
techGenStateId  
techStartGenLxNoOverlapLayer(  
    techFileId techfile  
);
```

Description

Starts the generator for the `LxNoOverlapLayer` function.

The function searches for the `virtuosoDefaultExtractorSetup` constraint group in the `techfile`. The function then searches the constraint group for `errorLayer` constraints. If the `errorLayer` constraint value is `TRUE`, the layer is retrieved. The function then determines whether the layer is a derived layer. If the layer is not a derived layer, the constraint is ignored.

If the layer is a derived layer, one of the following occurs.

- Layer1 and layer2 are returned from the derived layer. The purpose for both layers is `techcAllPurpose`.
- If the derived layer has an operation other than `AND`, the `errorLayer` constraint is ignored.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used with the <code>LxNoOverlapLayer</code> function. Returns a <code>NULL</code> generator state id, if the <code>virtuosoDefaultExtractorSetup</code> constraint group is not found.

techGenLxNoOverlapLayer

Boolean

```
techGenLxNoOverlapLayer(  
    techGenStateId  stateId,  
    techLayerNum    *layer1Num,  
    techPurpose     *purpose1,  
    techLayerNum    *layer2Num,  
    techPurpose     *purpose2  
);
```

Description

Generates the next set of two layer-purpose pairs.

Arguments

stateId	The generator state identifier returned from the techStartGenLxNoOverlapLayer function.
*pLayer1Num	The layer number of the first layer-purpose pair, returned by indirection.
*pPurpose1	The purpose number of the first layer-purpose pair, returned by indirection.
*pLayer2Num	The layer number of the second layer-purpose pair, returned by indirection.
*pPurpose2	The layer number of the second layer-purpose pair, returned by indirection.

Return Value

Boolean	Returns TRUE if the values for the four pointer arguments are assigned and valid. Returns FALSE if the generator state identifier is invalid or if the generator is finished.

techSetLxNoOverlapLayer

Boolean

```
techSetLxNoOverlapLayer(  
    techFileId      techfile,  
    techLayerNum    *layer1Num,  
    techPurpose     *purpose1,  
    techLayerNum    *layer2Num,  
    techPurpose     *purpose2  
);
```

Description

This function does the following.

- Creates a derived layer from the two specified layers.
- creates an errorLayer constraint with a value of TRUE on the derived layer.
- Adds the constraint to the `virtuosoDefaultExtractorSetup` constraint group.

If the constraint group does not exist, the function creates it.

If a derived layer with these two layers already exists, but the operation is different, the function creates a new derived layer.

If a derived layer with these two layers and the same operation already exists, but the value is FALSE, the value updates to TRUE.

Arguments

techfile	The database identifier of the technology file.
layer1Num	The layer number of the first layer-purpose pair.
purpose1	The purpose number of the first layer-purpose pair.
layer2Num	The layer number of the second layer-purpose pair.
purpose2	The purpose number of the second layer-purpose pair.

Return Value

Boolean	Returns FALSE if the technology file does not exist or if either or both layer-purpose pairs do not exist.

techIsLxNoOverlapLayer

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

Place and Route Rule Functions

Placement and routing rules are defined in constraint groups. The `LEFDefaultRouteSpec` constraint group in the technology file contains the constraints (the rules) for a design.

Note: The functions do not actively use the *purpose* argument. This includes layer purpose pairs. The *purpose* argument provides legacy compatibility with DFII only. Any *purpose* value supplied by the caller is stored by Virtuoso applications, but not used.

prRoutingLayers

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrRoutingLayer

```
techGenStateId  
techStartGenPrRoutingLayer(  
    techFileId techfile  
);
```

Description

Starts generating all the routing layers in the `validLayers` constraint in the `LEFDefaultRouteSpec` constraint group. A routing layer has a valid mask number and material set.

The generator obtains the preferred routing direction of each layer. If a layer does not have a preferred routing direction, the generator returns `none` and issues a warning.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used in the <code>GenPrRoutingLayer</code> function.

techGenPrRoutingLayer

Boolean

```
techGenPrRoutingLayer(  
    techGenStateId    stateId,  
    techLayerNum      *pLayerNum,  
    techPurpose       *pPurpose,  
    techPropType      *pType,  
    techPropValue     *pValue  
);
```

Description

Generates all the routing layers in the `validLayers` constraint in the `LEFDefaultRouteSpec` constraint group. A routing layer has a valid mask number and material set.

The generator obtains the preferred routing direction of each layer. If a layer does not have a preferred routing direction, the generator returns `none` and issues a warning.

Arguments

<code>stateId</code>	The generator state identifier returned by the <code>techStartGenPrRoutingLayer</code> function.
<code>*pLayerNum</code>	The layer number of the layer-purpose pair returned by the function.
<code>*pPurpose</code>	The purpose number of the layer-purpose pair returned by the function.
<code>*pType</code>	<i>pType</i> and <i>pValue</i> together specify the preferred routing direction of the layer. Returned by the function.
<code>*pValue</code>	<i>pType</i> and <i>pValue</i> together specify the preferred routing direction of the layer. Returned by the function.

Return Value

Boolean	Returns FALSE if the arguments are invalid or if the generator is finished.

techSetPrRoutingLayer

Boolean

```
techSetPrRoutingLayer(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Adds the `oaPhysicalLayer` of the `layerNum` to the `validLayers` constraint in the `LEFDefaultRouteSpec` constraint group. If a layer associated with the `layerNum` already exists, the function compares the preferred routing direction with the specified direction. If the direction is the same, the function returns a `TRUE`. If the direction is not the same, the function updates the direction and issues a warning.

If the `LEFDefaultRouteSpec` constraint group does not exist, the function creates it. A `validLayers` constraint containing this single layer is created and added to the constraint group.

If the `LEFDefaultRouteSpec` constraint group does exist, but the `validLayers` constraint does not exist within it, the function creates the `validLayers` constraint with this single layer.

In both cases, the function compares the preferred direction with the existing value either returning `TRUE` or updating the direction and issuing a warning.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

techfile	The database identifier of the technology file.
layerNum	The layer number of the layer-purpose pair.
purpose	The purpose number of the layer-purpose pair.
type	<i>Type</i> and <i>value</i> together specify the preferred routing direction.
value	<i>Type</i> and <i>value</i> together specify the preferred routing direction.

Return Value

Boolean	Returns TRUE if the preferred routing direction is the same as the existing value for the layer. If the preferred routing direction is not the same as the existing value, the function updates the direction and issues a warning. Returns FALSE if the technology file does not exist.

techGetPrRoutingDirection

Boolean

```
techGetPrRoutingDirection(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *pType,  
    techPropValue   *pValue  
);
```

Description

Gets the routing direction of the given layer.

The function does the following.

- Verifies the layer is a valid `oaPhysicalLayer` number.
- Finds the layer in the `validLayers` constraint in the `LEFDefaultRouteSpec` constraint group.
- Returns the preferred routing direction of the layer. If the layer has no preferred routing direction, returns `none` and issues a warning.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair.
<code>purpose</code>	The purpose number of the layer-purpose pair.
<code>*pType</code>	<i>pType</i> and <i>pValue</i> together specify the preferred routing direction. Returned by the function.
<code>*pValue</code>	<i>pType</i> and <i>pValue</i> together specify the preferred routing direction. Returned by the function.

Return Value

Boolean	Returns <code>FALSE</code> if the layer is not included in the <code>validLayers</code> constraint in the <code>LEFDefaultRouteSpec</code> constraint group.

techIsPrRoutingLayer

Boolean

```
techIsPrRoutingLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Verifies whether the `oaPhysicalLayer` of the `layerNum` is in the `validLayers` constraint in the `LEFDefaultRouteSpec` constraint group.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair.
<code>purpose</code>	The purpose number of the layer-purpose pair.

Return Value

Boolean	Returns FALSE if the layer is not included in the <code>validLayers</code> constraint of the <code>LEFDefaultRouteSpec</code> constraint group or if the technology file does not exist.

techGetPrRoutingPitch

Boolean

```
techGetPrRoutingPitch(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *pType,  
    techPropValue   *pValue  
);
```

Description

Gets the routing pitch of the specified layer. Routing pitch is the minimum spacing between two regular geometries on different nets.

The function does the following.

- Verifies the layer is a valid routing layer.
- Gets the preferred direction.
- Searches for the `oacHorizontalRoutingGridPitch` constraint or the `oacVerticalRoutingGridPitch` constraint depending on the preferred routing direction.
- If found, the function returns the routing pitch in the property. If no pitch is found, the function returns a 0.

You may add diagonal routing directions and the corresponding constraints.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair.
<code>purpose</code>	The purpose number of the layer-purpose pair.
<code>*pType</code>	<i>pType</i> and <i>pValue</i> together specify the routing pitch. Returned by the function.
<code>*pValue</code>	<i>pType</i> and <i>pValue</i> together specify the routing pitch. Returned by the function.

Return Value

Boolean	Returns FALSE if the routing pitch does not exist or if the technology file does not exist.

techSetPrRoutingPitch

Boolean

```
techSetPrRoutingPitch(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Sets the routing pitch for the specified layer. The function does the following.

- Verifies the layer as a valid routing layer.
- Get the preferred routing direction of the layer.
- Depending on the preferred routing direction, creates either the `oacHorizontalRoutingGridPitch` constraint or the `oacVerticalRoutingGridPitch` constraint with the specified value.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair.
<code>purpose</code>	The purpose number of the layer-purpose pair.
<code>type</code>	<i>type</i> and <i>value</i> together specify the preferred routing pitch.
<code>value</code>	<i>type</i> and <i>value</i> together specify the preferred routing pitch.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

techGetPrRoutingOffset

Boolean

```
techGetPrRoutingOffset(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purpose,  
    techPropType    *pType,  
    techPropValue   *pValue  
);
```

Description

Returns the routing offset for the specified layer. Routing offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids.

The function does the following.

- Verifies the layer is a valid routing layer.
- Gets the preferred direction.
- Searches for the `oacHorizontalRoutingGridOffset` constraint or the `oacVerticalRoutingGridOffset` constraint depending on the preferred routing direction.
- If found, the function returns the offset value in the property. If no offset is found, the function returns a 0.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

techfile	The database identifier of the technology file.
layerNum	The layer number of the layer-purpose pair.
purpose	The purpose number of the layer-purpose pair.
*pType	<i>pType</i> and <i>pValue</i> together specify the routing offset. Returned by the function.
*pValue	<i>pType</i> and <i>pValue</i> together specify the routing offset. Returned by the function.

Return Value

Boolean	Returns FALSE if the routing offset does not exist or if the technology file does not exist.

techSetPrRoutingOffset

Boolean

```
techSetPrRoutingOffset(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose      purpose,  
    techPropType     type,  
    techPropValue    value  
);
```

Description

Updates the routing offset for the specified layer. Routing offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids.

The function does the following.

- Verifies the layer as a valid routing layer.
- Get the preferred direction of the layer.
- Depending on the preferred routing direction, creates either the `oacHorizontalRoutingGridOffset` constraint or the `oacVerticalRoutingGridOffset` constraint with the specified value.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number of the layer-purpose pair.
<code>purpose</code>	The purpose number of the layer-purpose pair.
<code>type</code>	<i>type</i> and <i>value</i> together specify the routing offset.
<code>value</code>	<i>type</i> and <i>value</i> together specify the routing offset.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

prViaTypes

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrViaType

```
techGenStateId  
techStartGenPrViaType(  
    techFileId techfile  
);
```

Description

Starts generating all the viaDefs in the `validVias` constraint of all constraint groups.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used in the <code>techGenPrViaType</code> function.

techGenPrViaType

Boolean

```
techGenPrViaType (
    techGenStateId    stateId,
    String             *pDeviceName,
    String             *pDeviceView,
    techPropType       *pType,
    techPropValue      *pValue
);
```

Description

Generates all the viaDefs in the `validVias` constraint in all constraint groups.

Arguments

<code>stateId</code>	The generator state identifier returned by the <code>techStartGenPrViaType</code> function.
<code>*pDeviceName</code>	The viaDef name.
<code>*pDeviceView</code>	If the viaDef is custom, the view name of the cellview. If the viaDef is standard, the view name is <code>layout</code> .
<code>*pType</code>	<i>pType</i> and <i>pValue</i> together specify the name of the constraint group. Returned by the function.
<code>*pValue</code>	<i>pType</i> and <i>pValue</i> together specify the name of the constraint group. Returned by the function.

Return Value

Boolean	Returns FALSE if the arguments are invalid or if the generator is finished.

techSetPrViaType

Boolean

```
techSetPrViaType(  
    techFileId      techfile,  
    String          deviceName,  
    String          deviceView,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Adds the viaDef named *deviceName_via* to the `validVias` constraint in the specified constraint group if the following criteria are met.

- *deviceName_via* exists in the technology file.
- *deviceName_via* is a custom viaDef.
- *deviceView* is a fixed, meaning a non-p-cell, cellview.

If *deviceName_via* does not exist in the technology file, this viaDef is created and added to the `validVias` constraint.

If the constraint group and/or the `validVias` constraint does not exist, it is created and the viaDef is added.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>deviceName</code>	The viaDef name.
<code>deviceView</code>	If the viaDef is custom, the view name of the cellview. If the viaDef is standard, the view name is <code>layout</code> .
<code>type</code>	<i>type</i> and <i>value</i> together specify the name of the constraint group.
<code>value</code>	<i>type</i> and <i>value</i> together specify the name of the constraint group.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> and issues a warning if the viaDef is not custom or if the cellview is not a fixed cellview.

techGetPrViaType

Boolean

```
techGetPrViaType(  
    techFileId      techfile,  
    String          deviceName,  
    String          deviceView,  
    techPropType    *pType,  
    techPropValue   *pValue  
);
```

Description

Gets the constraint group name of the specified viaDef.

The function finds the viaDef named *deviceName_via* in the technology file. It then searches for the viaDef in the *validVias* constraint in all constraint groups in the technology file.

- If the viaDef is found, the function returns the constraint group name.
- If the constraint group is LEFDefaultRouteSpec, the functions returns *default* instead of the constraint group name.

If more than one constraint group has a *validVias* constraint containing the viaDef, the function returns the first occurrence.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>deviceName</code>	The <code>viaDef</code> name.
<code>deviceView</code>	If the <code>viaDef</code> is custom, the view name of the cellview. If the <code>viaDef</code> is standard, the view name is <code>layout</code> .
<code>*pType</code>	<i>pType</i> and <i>pValue</i> together specify the name of the constraint group. Returned by the function.
<code>*pValue</code>	<i>pType</i> and <i>pValue</i> together specify the name of the constraint group. Returned by the function.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the <code>viaDef</code> is not found in the technology file or is not a custom <code>viaDef</code> .

techIsPrViaDevice

Boolean

```
techIsPrViaDevice(  
    techFileId    techfile,  
    String        deviceName,  
    String        deviceView  
);
```

Description

Determines whether the specified device is a valid prVia. A valid prVia is a custom viaDef defined in the `validVias` constraint in the `LEFDefaultRouteSpec` constraint group.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>deviceName</code>	The viaDef name.
<code>deviceView</code>	If the viaDef is custom, the view name of the cellview. If the viaDef is standard, the view name is <code>layout</code> .

Return Value

Boolean	Returns FALSE if the device is not a custom via or if the technology file does not exist.

prOverlapLayers

The overlap layer supports older LEFIN/LEFOUT translators. For consistency with the lef2oa translator, a technology database hierarchical property named `LEF legacy` contains the name of the overlap layer as a string property. The property stores only one overlap layer. A layer set would replace the existing property with the specified name of the set.

techStartGenPrOverlapLayer

```
techGenStateId  
techStartGenPrOverlapLayer(  
    techFileId    techfile  
);
```

Description

Starts generating only the overlap layer stored as a string property on the `LEF legacy` property. `LEF legacy` is a technology database hierarchical property.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used in the <code>techGenPrOverlapLayer</code> function.

techGenPrOverlapLayer

Boolean

```
techGenPrOverlapLayer(  
    techGenStateI    stateId,  
    techLayerNum     *pLayerNum,  
    techPurpose      *pPurpose  
);
```

Description

Generates only the overlap layer stored as a string property on the `LEF legacy` property. `LEF legacy` is a technology database hierarchical property.

Arguments

<code>stateId</code>	The generator state identifier returned by the <code>techStartGenPrOverlapLayer</code> function.
<code>*pLayerNum</code>	The layer number of the first layer-purpose pair.
<code>*pPurpose</code>	The purpose number of the second layer-purpose pair.

Return Value

Boolean	Returns <code>FALSE</code> if the generator state identifier is invalid or if the generator is finished.

techIsPrOverlapLayer

Boolean

```
techIsPrOverlapLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Verifies whether the layer associated with the *layerNum* has the same name as the stored property.

Arguments

techfile	The database identifier of the technology file.
layerNum	The layer number.
purpose	The purpose number.

Return Value

Boolean	Returns FALSE if the layer does not have the same name as the stored property.

techSetPrOverlapLayer

Boolean

```
techSetPrOverlapLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Creates or replaces the property value with the given layer.

Arguments

techfile	The database identifier of the technology file.
layerNum	The layer number of the layer-purpose pair.
purpose	The purpose number of the layer-purpose pair.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

prViaStacks

`prViaStacks` indicate whether two layers can be stacked. The boolean value of the `oacViaStackAllowed` layer pair constraint in the `LEFDefaultRouteSpec` designates whether stacking is allowed. Stackable layers must be material `cut` or `li`, which is `contactlessMetal` on `OpenAccess`, and the layers must be different.

techStartGenPrStackVia

```
techGenStateId  
techStartGenPrStackVia(  
    techFileId techfile  
);
```

Description

Starts generating all the `oacViaStackAllowed` layer pair constraints having a value of `TRUE` in the `LEFDefaultRouteSpec` constraint group.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used in the <code>techGenPrStackVia</code> function.

techGenPrStackVia

Boolean

```
techGenPrStackVia(  
    techGenStateI    stateId,  
    techLayerNum     *pLayer1Num,  
    techPurpose      *pPurpose1,  
    techLayerNum     *pLayer2Num,  
    techPurpose      *pPurpose2  
);
```

Description

Generates all the `oacViaStackAllowed` layer pair constraints having a value of TRUE in the `LEFDefaultRouteSpec` constraint group.

Arguments

<code>stateId</code>	The generator state identifier returned by the <code>techStartGenPrStackVia</code> function.
<code>*pLayer1Num</code>	The layer number of the first layer-purpose pair.
<code>*pPurpose1</code>	The purpose number of the first layer-purpose pair.
<code>*pLayer2Num</code>	The layer number of the second layer-purpose pair.
<code>*pPurpose2</code>	The purpose number of the second layer-purpose pair.

Return Value

Boolean	Returns FALSE if the generator state identifier is invalid or if the generator is finished.

techSetPrStackVia

Boolean

```
techSetPrStackVia(  
    techFileId      techfile,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2  
);
```

Description

Creates an `oacViaStackAllowed` layer pair constraint with a value of TRUE for the two specified layers.

- If no `stackable` constraint in the `spacings` section of the techfile exists, this function creates it with the specified data.
- If the constraint already exists for the two layers, the constraint value updates to TRUE.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layer1Num</code>	The layer number of the first layer-purpose pair.
<code>purpose1</code>	The purpose number of the first layer-purpose pair.
<code>layer2Num</code>	The layer number of the second layer-purpose pair.
<code>purpose2</code>	The purpose number of the second layer-purpose pair.

Return Value

Boolean	Returns FALSE if both layers are not of material <code>cut</code> or <code>li</code> and each layer is different.

techIsPrStackVia

Boolean

```
techIsPrStackVia(  
    techFileId      techfile,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2  
);
```

Description

Determines whether the two layers have an `oacViaStackAllowed` layer pair constraint with a value of `TRUE` in the `LEFDefaultRouteSpec` constraint group.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layer1Num</code>	The layer number of the first layer-purpose pair.
<code>purpose1</code>	The purpose number of the first layer-purpose pair.
<code>layer2Num</code>	The layer number of the second layer-purpose pair.
<code>purpose2</code>	The purpose number of the second layer-purpose pair.

Return Value

Boolean	Returns <code>FALSE</code> if the layers do not have an <code>oacViaStackAllowed</code> layer pair constraint with a value of <code>TRUE</code> in the <code>LEFDefaultRouteSpec</code> constraint group.

prViaMaxStackVias

`prViaMaxStackVias` indicates the maximum stack between two layers. The integer value of the `viaStackingLimit` constraint in the constraint group is the maximum stack allowed. Specifying layers is optional. Specifying layers allows more than one `viaStackingLimit` constraint in the same constraint group. Otherwise, one `viaStackingLimit` applies to all layer pairs in the constraint group.

techStartGenPrMaxStackVia

```
techGenStateId  
techStartGenPrMaxStackVia(  
    techFileId techfile  
);
```

Description

Starts generating all the `viaStackingLimit` constraints in the `LEFDefaultRouteSpec` constraint group. The integer value is the limit between the two layers.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used in the <code>techGenPrStackVia</code> function.

techGenPrMaxStackVia

Boolean

```
techGenPrMaxStackVia(  
    techGenStateId    stateId,  
    techPropType      *pType,  
    techPropValue     *pValue,  
    techLayerNum      *pLayer1Num,  
    techPurpose       *pPurpose1,  
    techLayerNum      *pLayer2Num,  
    techPurpose       *pPurpose2  
);
```

Description

Generates all the `viaStackingLimit` constraints in the `LEFDefaultRouteSpec` constraint group. The integer value is the limit between the two layers.

Arguments

<code>stateId</code>	The generator state identifier returned by the <code>techStartGenPrStackVia</code> function.
<code>*pType</code>	<i>*pType</i> and <i>*pValue</i> together specify the maximum stacking limit between the layers.
<code>*pValue</code>	<i>*pValue</i> and <i>*pType</i> together specify the maximum stacking limit between the layers.
<code>*pLayer1Num</code>	The layer number of the first layer-purpose pair. Optional. If not specified, the stacking limit applies to all layers.
<code>*pPurpose1</code>	The purpose number of the first layer-purpose pair. Optional. If not specified, the stacking limit applies to all layers.
<code>*pLayer2Num</code>	The layer number of the second layer-purpose pair. Optional. If not specified, the stacking limit applies to all layers.
<code>*pPurpose2</code>	The purpose number of the second layer-purpose pair. Optional. If not specified, the stacking limit applies to all layers.

Return Value

Boolean	Returns FALSE if the generator state identifier is invalid or if the generator is finished.

techSetPrMaxStackVia

Boolean

```
techSetPrMaxStackVia (
    techPropType    type,
    techPropValue   value,
    techFileId      techfile,
    techLayerNum    layer1Num,
    techPurpose     purpose1,
    techLayerNum    layer2Num,
    techPurpose     purpose2
);
```

Description

Sets the `viaStackingLimit` between the two layers. The function searches for the `viaStackingLimit` constraint on the two specified layers in the `LEFDefaultRouteSpec` constraint group. If found, the function updates the value. If the `viaStackingLimit` constraint does not exist, the function creates the constraint and adds it to the `LEFDefaultRouteSpec` constraint group.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>type</code>	<i>type</i> and <i>value</i> together specify the maximum stacking limit between the layers.
<code>value</code>	<i>value</i> and <i>type</i> together specify the maximum stacking limit between the layers.
<code>layer1Num</code>	The layer number of the first layer-purpose pair.
<code>purpose1</code>	The purpose number of the first layer-purpose pair.
<code>layer2Num</code>	The layer number of the second layer-purpose pair.
<code>purpose2</code>	The purpose number of the second layer-purpose pair.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

prMastersliceLayers

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrMastersliceLayer

```
techGenStateId  
techStartGenPrMastersliceLayer(  
    techFileId techfile  
);
```

Description

Starts generating all the masterslice layers in the technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used with the <code>techGenPrMastersliceLayer</code> function.

techGenPrMastersliceLayer

Boolean

```
techGenPrMastersliceLayer (  
    techGenStateId    stateId,  
    techLayerNum      *pLayerNum,  
    techPurpose       *pPurpose  
);
```

Description

Generates all the masterslice layers in the technology file.

Arguments

stateId	The state identifier returned by the techStartGenPrMastersliceLayer function.
*pLayerNum	The layer number of the layer-purpose pair. Returned by the function.
*pPurpose	The purpose number of the layer-purpose pair. Returned by the function.

Return Value

Boolean	Returns FALSE if the generator state identifier is invalid or if the generator is finished.

techSetPrMastersliceLayer

Boolean

```
techSetPrMastersliceLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Determines whether the layer associated with the `layerNum` is a masterslice layer.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number.
<code>purpose</code>	The purpose number.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

techIsPrMastersliceLayer

Boolean

```
techIsPrMastersliceLayer(  
    techFileId    techfile,  
    techLayerNum  layerNum,  
    techPurpose   purpose  
);
```

Description

Determines if the layer associated with the `layerNum` is a masterslice layer in the specified technology file.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>layerNum</code>	The layer number.
<code>purpose</code>	The purpose number.

Return Value

Boolean	Returns FALSE if the layer is found in the <code>validLayers</code> constraint of the <code>LEFDefaultRouteSpec</code> constraint group or if the layer is not one of the following materials: <code>nwell</code> , <code>pwell</code> , <code>ndiff</code> , <code>pdiff</code> , or <code>poly</code> .

prViaRules

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrViaRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techGenPrViaRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techSetPrViaRule

Boolean

```
techSetPrViaRule(  
    techFileId          techfile,  
    String              ruleName,  
    techParamStruct     *pViaTypes,  
    int                 numberOfVias,  
    techLayerNum        layer1Num,  
    techPurpose         purpose1,  
    techPropType        dir1Type,  
    techPropValue       dir1Value,  
    techParamStruct     *pParams1,  
    techLayerNum        layer2Num,  
    techPurpose         purpose2,  
    techPropType        dir2Type,  
    techPropValue       dir2Value,  
    techParamStruct     *pParams2  
);
```

Description

Sets the prViaRule in the technology file.

If a viaSpec already exists, the function ensures the layers match.

If the layers do not match, the function aborts and issues a warning.

If the layers do match, the function updates the direction of the layers and issues a warning if the value specified differs from the existing value.

The function gets the viaDef2DTbl from the viaSpec.

If no table exists, a table is created as follows.

- ❑ the viaDef2DTbl is created with the width specified in the parameters.
- ❑ the viaDefs are added in a viaDef array and set in the viaDef2DTbl.

If the table exists, the function searches the table header for the layer width values.

- ❑ if the layer width values are found, the function updates the indexes of the viaDefArray entry to include the viaDefs specified, if the viaDefs are not already in the array.
- ❑ if the layer width values are not found, the function inserts the width value into the table and adds the corresponding viaDefArray entry value.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

If a viaSpec does not already exist, the function creates a new viaSpec with the specified layers. It also creates a viaDef2DTbl with the values specified in the headers.

Arguments

techfile	The database identifier of the technology file.
ruleName	The name of the via rule. Ignored if a via spec for the layers already exists and has a name.
*pViaTypes	The array of via types.
numberOfVias	The number of vias.
layer1Num	The layer number of the first layer-purpose pair. Ignored. Layers are defined in the technology file.
purpose1	The purpose number of the first layer-purpose pair. Ignored. Purpose provides compatibility with Virtuoso applications only.
dir1Type	<i>dir1Type</i> and <i>dir1Value</i> together specify the direction of the first layer-purpose pair.
dir1Value	<i>dir1Value</i> and <i>dir1Type</i> together specify the direction of the first layer-purpose pair.
*pParams1	The array of parameters for the first layer-purpose pair.
layer2Num	The layer number of the second layer-purpose pair. Ignored. Layers are defined in the technology file.
purpose2	The purpose number of the second layer-purpose pair. Ignored. Purpose provides compatibility with Virtuoso applications only.
dir2Type	<i>dir2Type</i> and <i>dir2Value</i> together specify the direction of the second layer-purpose pair.
dir2Value	<i>dir2Value</i> and <i>dir2Type</i> together specify the direction of the second layer-purpose pair.
*pParams2	The array of parameters for the second layer-purpose pair.

Return Value

Boolean	Returns FALSE if the technology file does not exist.

prGenViaRules

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrGenViaRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techSetPrGenViaRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techGen

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrTurnViaRule

Note: This function is ignored.

```
techGenStateId  
techStartGenPrTurnViaRule(  
    techFileId techfile  
);
```

Description

Starts the generator to generate the turn via rule in the Place and Route rules class in the technology file.

Arguments

techfile	The database identifier of the technology file.

Return Value

techGenStateId	Returns the generator state identifier to be used with the GenPrTurnViaRule function.

techGenPrTurnViaRule

Note: This function is ignored.

Boolean

```
techGenPrTurnViaRule(  
    techGenStateId    stateId,  
    String             *pRuleName,  
    techLayerNum       *pLayerNum,  
    techPurpose        *pPurpose,  
    techPropType       *pDir1Type,  
    techPropValue      *pDir1Value,  
    techPropType       *pDir2Type,  
    techPropValue      *pDir2Value  
);
```

Description

Generates the parameters for the specified turn via rule in the `prTurnViaRules` rules of the specified technology file.

Arguments

<code>stateId</code>	The state identifier returned by the <code>GenPrTurnViaRule</code> function.
<code>*pRuleName</code>	The turn via rule name.
<code>*pLayerNum</code>	The layer number of the layer-purpose pair. Returned by the function.
<code>*pPurpose</code>	The purpose number of the layer-purpose pair. Returned by the function.
<code>*pDir1Type</code>	The type of the first direction.
<code>*pDir1Value</code>	The value of the first direction.
<code>*pDir2Type</code>	The type of the second direction.
<code>*pDir2Value</code>	The value of the second direction.

Return Value

Boolean	Returns FALSE if the generator state identifier is not valid, if the technology file does not exist, or if the generator is finished.

prNonDefaultRules

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrNonDefaultRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techGenPrNonDefaultRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techSetPrNonDefaultRule

Note: This function is no longer supported and will be removed in the upcoming releases. Cadence recommends that you remove it from your technology files.

techStartGenPrNonDefaultParam

```
techRuleGenStateId  
techStartGenPrNonDefaultParam  
    techClassId  ruleId  
    );
```

Description

Starts generating the parameters of the non-default rule.

Arguments

<code>ruleId</code>	The identifier of the rule to be used in the <code>GenPrNonDefaultParam</code> function.

Return Value

<code>techRuleGenStateId</code>	Returns the generator state identifier to be used in the <code>techGenPrNonDefaultParam</code> function.

techGenPrNonDefaultParam

Boolean

```
techGenPrNonDefaultParam(  
    techRuleGenStateId    ruleStateId,  
    techLayerNum          *pLayerNum,  
    techPurpose           *pPurpose,  
    techParamStruct       *pParams,  
    String                *pViaName,  
    techLayerNum          *pViaLayer1Num,  
    techPurpose           *pViaPurpose1,  
    techLayerNum          *pViaLayer2Num,  
    techPurpose           *pViaPurpose2,  
    techParamStruct       *pStackViaParams  
);
```

Description

Generates the parameters of the non-default rule. The class id is the constraint group id. The function generates layers and viaDefs if the constraint group has `validLayers` and `validVias` constraints.

The `stackVia` comes from the stacking constraints in the constraint group. The function fills constraint values in the `stackViaParams` if it finds stacking constraints. If the constraints do not exist, the function uses `NULL`.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>ruleStateId</code>	The generator state identifier returned from the <code>StartGenPrNonDefaultParam</code> rule.
<code>*pLayerNum</code>	The layer number of the layer-purpose pair. Returned by the function.
<code>*pPurpose</code>	The purpose number of the layer-purpose pair. Returned by the function.
<code>*pParams</code>	The array of parameters. Returned by the function.
<code>*pViaName</code>	The via name. Returned by the function.
<code>*pViaLayer1Num</code>	The via layer number for the first layer-purpose pair. Returned by the function.
<code>*pViaPurpose1</code>	The via purpose number for the first layer-purpose pair. Returned by the function.
<code>*pViaLayer2Num</code>	The layer number for the second layer-purpose pair. Returned by the function.
<code>*pViaPurpose2</code>	The purpose number for the second layer-purpose pair. Returned by the function.
<code>*pStackViaParams</code>	The array of via parameters. Returned by the function.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the <code>validLayers</code> constraint or the <code>validVias</code> constraint does not exist.

viaDef Functions

This section describes functions affecting viaDef objects.

About viaDefs

A viaDef object defines a via cellview in a technology database. A via in a route must point to a viaDef object in the technology database associated with the cellview where the route resides. There are two types of viaDefs, custom viaDef and standard viaDef.

techList2ViaDef

```
techViaDefId  
techList2ViaDef(  
    String      funcName,  
    list        l  
);
```

Description

Gets the corresponding viaDef according to the list passed in.

Arguments

<i>funcName</i>	The name of the calling function.
<i>l</i>	The list passed in.

Return Values

techViaDefId	The viaDef database id.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techIsViaDefId

Boolean

```
techIsViaDefId(  
    techViaDefId    viaDef  
);
```

Description

Checks that the given *viaDef* refers to a viaDef object.

Arguments

<i>viaDef</i>	The database techViaDefId for the viaDef object.

Return Values

Boolean	Returns TRUE if the <i>viaDef</i> refers to a viaDef object. Returns FALSE if it is not a viaDef object.

techIsCustomViaDefId

Boolean

```
techIsCustomViaDefId(  
    techViaDefId    viaDef  
);
```

Description

Checks that the given *viaDef* refers to a customViaDef object.

Arguments

<i>viaDef</i>	The database <code>techViaDefId</code> for the customViaDef object.

Return Values

Boolean	Returns TRUE if the <i>viaDef</i> refers to a customViaDef object. Returns FALSE if it is not a customViaDef object.

techIsStandardViaDefId

Boolean

```
techIsStandardViaDefId(  
    techViaDefId    viaDef  
);
```

Description

Checks that the given *viaDef* refers to a standardViaDef object.

Arguments

<i>viaDef</i>	The database techViaDefId for the standardViaDef object.

Return Values

Boolean	Returns TRUE if the <i>viaDef</i> refers to a standardViaDef object. Returns FALSE if it is not a standardViaDef object.

techFindViaDefByName

```
techViaDefId  
techFindViaDefByName(  
    techFileId    techId,  
    String        name  
);
```

Description

Searches the technology file specified in *techId* for a viaDef object with the specified name.

Arguments

<i>techId</i>	The database <code>techFileId</code> of the technology file.
<i>name</i>	The name of the viaDef object to be searched for.

Return Values

<code>techViaDefId</code>	The database id of the techViaDef object found.

techDeleteViaDef

Boolean

```
techDeleteViaDef(  
    techViaDefId    viaDefId  
);
```

Description

Deletes the viaDef object from the specified technology file.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

Boolean	Returns TRUE if the viaDef object was successfully deleted. Returns FALSE otherwise.

techGetViaDefName

```
String  
techGetViaDefName(  
    techViaDefId    viaDefId  
);
```

Description

Gets the name of the specified viaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

String	The name of the viaDef object.

techCreateCustomViaDef

```
techViaDefId
techCreateCustomViaDef(
    techFileId      techId,
    String           name,
    dbCellViewId    masterId,
    techLayerId     layer1,
    techLayerId     layer2
);
```

Description

Creates a custom viaDef with the specified attributes in the viaDef section of the specified technology file.

Arguments

<i>techId</i>	The database id of the technology file.
<i>name</i>	The name to be assigned to the viaDef.
<i>masterId</i>	The database id of the master cellview. To be valid, the master should have the cellview cellType attribute set to <code>via</code> .
<i>layer1</i>	The first (bottom) physical layer, of the viaDef. A viaDef layer must be associated with a mask number to be valid.
<i>layer2</i>	The second (top) physical layer, of the viaDef. A viaDef layer must be associated with a mask number to be valid.

Return Values

<i>techViaDefId</i>	The database id of the techViaDef created.

techCreateCustomViaDefByName

```
techViaDefId
techCreateCustomViaDefByName (
    techFileId      techId,
    String           viaDefName,
    String           libName,
    String           cellName,
    String           viewName,
    techLayerId      layer1,
    techLayerId      layer2
);
```

Description

Creates a viaDef with the specified attributes in the viaDef section of the specified technology file. The master cellview is specified by its library, cell, and view names.

Arguments

<i>techId</i>	The database id of the technology file in which to create the customViaDef.
<i>viaDefName</i>	A string representing the customViaDef name. The name must be unique in a technology database.
<i>libName</i>	The libraryName of the master cellview.
<i>cellName</i>	The cellName of the master cellview.
<i>viewName</i>	The viewName of the master cellview.
<i>layer1</i>	The first (bottom) physical layer, of the viaDef. A viaDef layer must be associated with a mask number to be valid.
<i>layer2</i>	The second (top) physical layer, of the viaDef. A viaDef layer must be associated with a mask number to be valid.

Return Values

techViaDefId	The database id of the techViaDef created.

techGetCustomViaDefLibName

String

```
techGetCustomViaDefLibName(  
    techViaDefId    viaDefId  
);
```

Description

Gets the library name of the master cellview that is associated with the specified customViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

String	The library name of the master cellview that is associated with the specified customViaDef object.

techGetCustomViaDefCellName

```
String  
techGetCustomViaDefCellName (  
    techViaDefId      viaDefId  
);
```

Description

Gets the cell name of the master cellview that is associated with the specified customViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

String	The cell name of the master cellview that is associated with the specified customViaDef object.

techGetCustomViaDefViewName

String

```
techGetCustomViaDefViewName (  
    techViaDefId      viaDefId  
);
```

Description

Gets the view name of the master cellview that is associated with the specified customViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

String	The view name of the master cellview that is associated with the specified customViaDef object.

techGetViaDefLayers

Boolean

```
techGetViaDefLayers (  
    techViaDefId    viaDefId,  
    techLayerId     *layer1,  
    techLayerId     *layer2  
);
```

Description

Gets the layers associated with the specified viaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.
<i>*layer1</i>	The first (bottom) physical layer, of the viaDef. A viaDef layer must be associated with a mask number to be valid.
<i>*layer2</i>	The second (top) physical layer, of the viaDef. A viaDef layer must be associated with a mask number to be valid.

Return Values

Boolean	Returns TRUE if the viaDef object layers were successfully retrieved. Returns FALSE otherwise.

techGetViaDefLayer1

```
techLayerId  
techGetViaDefLayer1(  
    techViaDefId    viaDefId  
);
```

Description

Gets physical layer1 (bottom) associated with the specified viaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

<i>techLayerId</i>	The database id of physical layer1 (bottom) associated with the viaDef object.

techGetViaDefLayer1Num

```
techLayerNum  
techGetViaDefLayer1Num(  
    techViaDefId    viaDefId  
);
```

Description

Gets the layer number of physical layer1 (bottom) associated with the specified `viaDef` object.

Arguments

<i>viaDefId</i>	The database id of the <code>viaDef</code> object.

Return Values

<code>techLayerNum</code>	The layer number of physical layer1 (bottom) associated with the <code>viaDef</code> object.

techGetViaDefLayer2Num

```
techLayerNum  
techGetViaDefLayer1Num(  
    techViaDefId    viaDefId  
);
```

Description

Gets the layer number of physical layer2 (top) associated with the specified viaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

<i>techLayerNum</i>	The layer number of physical layer2 (top) associated with the viaDef object.

techGetViaDefLayer2

```
techLayerId  
techGetViaDefLayer2(  
    techViaDefId    viaDefId  
);
```

Description

Gets physical layer2 (top) associated with the specified viaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef object.

Return Values

<i>techLayerId</i>	The database id of physical layer2 (top) associated with the viaDef object.

techViaDefHasLayer

Boolean

```
techViaDefHasLayer(  
    techViaDefId      viaDefId,  
    techLayerNum      layerNum  
);
```

Description

Check whether a viaDef object is associated with the layer with the specified layer number.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef object.
<i>layerNum</i>	The layer number of the layer associated with the viaDef.

Return Values

Boolean	Returns TRUE if the viaDef object is associated with the layer with the specified layer number. Returns FALSE otherwise.

techGetViaDefResistancePerCut

```
float  
techGetViaDefResistancePerCut(  
    techViaDefId    viaDefId  
);
```

Description

Gets the `resistancePerCut` parameter value from the `viaDef`.

Arguments

<i>viaDefId</i>	The database id of the technology file <code>viaDef</code> .

Return Values

<code>float</code>	The <code>resistancePerCut</code> parameter value of the <code>viaDef</code> .

techSetViaDefResistancePerCut

Boolean

```
techSetViaDefResistancePerCut(  
    techViaDefId    viaDefId,  
    float           resPerCut  
);
```

Description

Sets the resistancePerCut attribute on this viaDef.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.
<i>resPerCut</i>	The resistance per cut shape value to be set for the viaDef.

Return Values

Boolean	Returns TRUE if the resistance per cut shape value is successfully set for the viaDef. Returns FALSE otherwise.

techGetNumViaDefs

```
int  
techGetNumViaDefs(  
    techFileId    techId  
);
```

Description

Gets the number of viaDefs in the specified techfile.

Arguments

<i>techId</i>	The database id of the technology file.

Return Values

<i>int</i>	The number of viaDefs in the specified techfile.

techStartGenViaDef

```
techGenStateId  
techStartGenViaDef(  
    techFileId    techId  
);
```

Description

Starts generating all the viaDefs in the given techfile.

Arguments

<i>techId</i>	The database id of the technology file.

Return Values

<i>techGenStateId</i>	Returns the generator <code>techGenStateId</code> used to track the state of viaDef generation.

techGetStdViaDefImplant2Num

```
techLayerNum  
techGetStdViaDefImplant2Num(  
    techViaDefId    viaDefId  
);
```

Description

Gets the layer number of the `implant2` layer associated with the specified `standardViaDef` object.

Arguments

<code>viaDefId</code>	The database id of the <code>standardViaDef</code> object.

Return Values

<code>techLayerNum</code>	The layer number of the <code>implant2</code> layer associated with the <code>standardViaDef</code> object. If an <code>implant2</code> layer does not exist, returns the <code>techcNullLayer</code> layer number (195).

techGenViaDef

Boolean

```
techGenViaDef(  
    techGenStateId    pState,  
    techViaDefId      *viaDefId  
);
```

Description

Generates the next viaDef in the techfile.

Arguments

<i>pState</i>	The generator <code>techGenStateId</code> used to track the state of viaDef generation.
<i>*viaDefId</i>	The <code>techViaDefId</code> pointer. Returned by the function if a viaDef is generated.

Return Values

Boolean	Returns TRUE if a viaDef is generated. Returns FALSE otherwise.

techCreateStdViaDef

```
techViaDefId  
techCreateStdViaDef(  
    techFileId      techId,  
    String           viaDefName,  
    techLayerId     layer1,  
    techLayerId     layer2,  
    dbViaParam      *param,  
    techLayerId     implant1,  
    techLayerId     implant2  
);
```

Description

Creates a standard viaDef with the specified attributes in the viaDef section of the specified technology file.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<i>techId</i>	The database id of the technology file.
<i>viaDefName</i>	The name to be assigned to the viaDef.
<i>*param</i>	The parameters for the stdViaDef object, as below. <pre>typedef void* dbViaCutPatternInfo; typedef struct dbViaParam { dbPoint layer1Enc; dbPoint layer2Enc; dbPoint imp1Enc; dbPoint imp2Enc; dbPoint layer1Offset; dbPoint layer2Offset; dbPoint cutSpacing; dbPoint originOffset; unsigned int cutWidth; unsigned int cutHeight; unsigned int cutLayer; unsigned int cutColumns; unsigned int cutRows; dbViaCutPatternInfo cutPatternInfo; } dbViaParam;</pre>
<i>layer1</i>	Layer1, the first (bottom) physical layer, of the stdViaDef object.
<i>layer2</i>	Layer2, the second (top) physical layer, of the stdViaDef object.
<i>implant1</i>	The optional implant1 layer of the stdViaDef object. If no implant1 layer is associated with this standard viaDef, passes NULL as the value.
<i>implant2</i>	The optional implant2 layer of the stdViaDef object. If no implant2 layer is associated with this standard viaDef, passes NULL as the value.

Return Values

<i>techViaDefId</i>	The database id of the techViaDef created. Otherwise techNullViaDefId is returned. An exception is thrown if a viaDef with the specified name already exist.

techStdViaDefHasImplant1

Boolean

```
techStdViaDefHasImplant1(  
    techViaDefId    viaDefId  
);
```

Description

Checks whether or not a `standardViaDef` object has an `implant1` layer.

Arguments

<code>viaDefId</code>	The database id of the <code>standardViaDef</code> object.

Return Values

Boolean	Returns TRUE if the <code>standardViaDef</code> object has an <code>implant1</code> layer. Returns FALSE otherwise.

techStdViaDefHasImplant2

Boolean

```
techStdViaDefHasImplant2(  
    techViaDefId    viaDefId  
);
```

Description

Checks whether or not a standard `viaDef` object has an `implant2` layer.

Arguments

<code>viaDefId</code>	The database id of the <code>standardViaDef</code> object.

Return Values

Boolean	Returns TRUE if the <code>standardViaDef</code> object has an <code>implant2</code> layer. Returns FALSE otherwise.

techGetStdViaDefImplant1

```
techLayerId  
techGetStdViaDefImplant1(  
    techViaDefId    viaDefId  
);
```

Description

Gets the implant1 layer (bottom layer implant) associated with the stdViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

<i>techLayerId</i>	The database id of the implant1 layer associated with the stdViaDef object. If the stdViaDef object does not have an associated implant1 layer, NULL is returned.

techGetStdViaDefImplant2

```
techLayerId  
techGetStdViaDefImplant2(  
    techViaDefId    viaDefId  
);
```

Description

Gets the implant2 layer (top layer implant) associated with the stdViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

<i>techLayerId</i>	The database id of the implant2 layer associated with the stdViaDef object. If the stdViaDef object does not have an associated implant1 layer, NULL is returned

techGetStdViaDefParams

```
dbViaParam*  
techGetStdViaDefParams(  
    techViaDefId    viaDefId  
);
```

Description

Gets the standard via parameters associated with the stdViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.

Return Values

<i>dbViaParam*</i>	A pointer pointing to the standard via parameter structure associated with the stdViaDef object.

techSetStdViaDefParams

Boolean

```
techSetStdViaDefParams(  
    techViaDefId    viaDefId,  
    dbViaPram      *params  
);
```

Description

Sets the standard via parameters associated with the stdViaDef object.

Arguments

<i>viaDefId</i>	The database id of the technology file viaDef.
<i>*params</i>	A pointer pointing to the standard via parameter structure associated with the stdViaDef object.

Return Values

Boolean	Returns TRUE if the via parameters were successfully set for the stdViaDef object. Returns FALSE otherwise.

viaSpec Functions

This section describes functions affecting viaSpec objects.

About Via Specifications

Via specification objects define arrays of via definitions.

techGetViaSpecDefaultViaDefNames

Boolean

```
techGetViaSpecDefaultViaDefNames (
    techViaSpecId      viaSpecId,
    unsigned int        *numViaDefs,
    string              **viaDefNames
);
```

Description

Gets the names of the `viaDefs` of the default value in a specific `viaSpec`. The value of the pointers is valid only if the function returns TRUE.

Arguments

<code>viaSpecId</code>	The database id of the <code>viaSpec</code> .
<code>*numViaDefs</code>	The pointer to the number of via definitions in the <code>viaSpec</code> default value.
<code>**viaDefNames</code>	The pointer to the list of names of the via definitions in the <code>viaSpec</code> default value.

Return Values

Boolean	Returns TRUE if the function successfully gets the <code>viaDef</code> names. Returns FALSE otherwise.

viaHeader Functions

This section describes functions affecting viaHeader objects.

About viaHeaders

Via Header objects hold information that is common to all vias referring to a specific viaHeader.

dbIsViaHeaderId

Boolean

```
dbIsViaHeaderId(  
    dbId      objId  
);
```

Description

Checks that the given objId is a valid a viaHeader object.

Arguments

<i>objId</i>	The database dbId for any database object.

Return Values

Boolean	Returns TRUE if the <i>objId</i> is a valid viaHeader object. Returns FALSE if it is not a viaHeader object.

dbFindViaHeaderByName

Boolean

```
dbFindViaHeaderByName (
    dbCellViewId      cvId,
    String             viaDefName,
    dbViaHeaderId*     viaHeaderId
);
```

Description

Searches the cellview specified by *cvId* for a super viaHeader object associated with a viaDef with the specified name.

Arguments

<i>cvId</i>	The database <i>cvId</i> of the cell file.
<i>viaDefName</i>	The name of the viaDef.
<i>viaHeaderId</i>	A pointer to the database <i>dbViaHeaderId</i> to be populated.

Return Values

Boolean	Returns TRUE if the viaHeader object was successfully found. Returns FALSE otherwise.

dbFindViaHeader

Boolean

```
dbFindViaHeaderByName (
    dbCellViewId      cvId,
    techViaDefId       viaDefId,
    dbViaHeaderId*     viaHeaderId
);
```

Description

Searches the cellview specified by *cvId* for a viaHeader object associated with a viaDef with the specified *viaDefId*.

Arguments

<i>cvId</i>	The databaseId of the cellview.
<i>viaDefId</i>	The technology id of the viaDef.
<i>viaHeaderId</i>	A pointer to the database dbViaHeaderId to be populated.

Return Values

Boolean	Returns TRUE if the viaHeader object was successfully found. Returns FALSE otherwise.

dbIsViaHeaderBound

Boolean

```
dbIsViaHeaderBound(  
    dbViaHeaderId    viaHeaderId  
);
```

Description

Checks whether specified the viaHeader is bound.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.

Return Values

Boolean	Returns TRUE if the viaHeader object is bound. Returns FALSE otherwise.

dbGetViaHeaderViaDef

Boolean

```
dbGetViaHeaderViaDef(  
    dbViaHeaderId      viaHeaderId,  
    techViaDefId*      viaDefId  
);
```

Description

Gets the technology database viaDef associated with the specified viaHeader object in the viaDefId parameter.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.
<i>viaDefId</i>	A pointer to the database techViaDefId to be populated.

Return Values

Boolean	Returns TRUE if the viaHeader object was successfully retrieved. Returns FALSE otherwise.

dbGetViaHeaderViaDefName

Boolean

```
dbGetViaHeaderViaDefName (
    dbViaHeaderId      viaHeaderId,
    String*             viaDefName
);
```

Description

Gets the name of the technology database viaDef associated with the specified viaHeader object in the viaDefName parameter. The viaHeader may or may not be bound.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.
<i>viaDefName</i>	A pointer to the name of the viaDef object to be populated.

Return Values

Boolean	Returns TRUE if the viaDefName was successfully populated. Returns FALSE otherwise.

dbStartGenViaHeader

```
dbGenStateId  
dbStartGenViaHeader(  
    dbCellViewId    cvId  
)
```

Description

Starts generating all the viaHeaders in the given cellview.

Arguments

<i>cvId</i>	The cellview id in which to generate the viaHeader objects.

Return Values

<i>dbGenStateId</i>	The generator <code>dbGenStateId</code> used to track the state of viaHeader object generation.

dbGenViaHeader

Boolean

```
dbGenViaHeader(  
    dbGenStateId      pState,  
    dbViaHeaderId*    viaHeaderId  
)
```

Description

Generates the next viaHeader in the cellview.

Arguments

<code>pState</code>	The generator <code>dbGenStateId</code> used to track the state of viaHeader object generation.
<code>viaHeaderId</code>	Pointer to be filled with the viaHeader in the cellview.

Return Values

Boolean	Returns TRUE if the viaHeader is generated. Returns FALSE otherwise.

dbGetViaHeaderMaster

Boolean

```
dbGetViaHeaderMaster(  
    dbViaHeaderId      viaHeaderId,  
    dbCellViewId*      cellViewId  
);
```

Description

Gets the cellviewId of the specified viaHeader's master cellview in the cellviewId parameter. The viaHeader must be bound.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.
<i>cellViewId</i>	A pointer to the cellview id of the viaDef object to be populated.

Return Values

Boolean	Returns TRUE if the cellview id of the viaHeader master cellview was successfully retrieved. Returns FALSE otherwise.

dbGetViaHeaderMasterBBox

Boolean

```
dbGetViaHeaderMasterBBox(  
    dbViaHeaderId    viaHeaderId,  
    dbBBox*          bBox  
);
```

Description

Populates the bBox parameter with the specified viaHeader's master cellview bBox.

Note: If the master bBox is inverted (possible if the master cellview is empty), the bBox parameter is set to zero in the returned result.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.
<i>bBox</i>	A pointer to the bBox data to be populated.

Return Values

Boolean	Returns TRUE if the bBox of the viaHeader master cellview was successfully populated into the bBox parameter. Returns FALSE otherwise.

dbIsViaSuperHeader

Boolean

```
dbIsViaSuperHeader(  
    dbViaHeaderId      viaHeaderId  
);
```

Description

Checks if the viaHeader object specified by viaHeaderId is a via superHeader

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.

Return Values

Boolean	Returns TRUE if the viaHeader object specified by <i>viaHeaderId</i> is a super viaHeader. Returns FALSE otherwise.

dbIsViaSubHeader

Boolean

```
dbIsViaSubHeader(  
    dbViaHeaderId    viaHeaderId  
);
```

Description

Checks if the viaHeader object specified by viaHeaderId is a via subHeader

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the viaHeader object.

Return Values

Boolean	Returns TRUE if the viaHeader object specified by <i>viaHeaderId</i> is a via subHeader. Returns FALSE otherwise.

dbStartGenViaSuperHeaderToSubHeader

```
dbGenStateId  
dbStartGenViaSuperHeaderToSubHeader (  
    dbViaHeaderId      viaSuperHeaderId  
);
```

Description

Starts generating the via subHeaders for the via superHeader.

Arguments

<i>viaSuperHeaderId</i>	The database id of the via superHeader.

Return Values

dbGenStateId	The generator dbGenStateId used to track the state of via subHeader generation.

dbGenViaSuperHeaderToSubHeader

Boolean

```
dbGenViaSuperHeaderToSubHeader(  
    dbGenStateId      pState,  
    dbViaHeaderId*    viaSubHeaderId  
);
```

Description

Generates the via subHeaders for a viaSuperHeader.

Arguments

<i>pstate</i>	The generator dbGenStateId used to track the state of viaSubHeader generation.
<i>viaSubHeaderId</i>	A pointer to the database id of the via subHeader.

Return Values

Boolean	Returns TRUE if the via subHeaders are generated successfully. Returns FALSE otherwise.

dbGetStdViaHeaderParams

Boolean

```
dbGetStdViaHeaderParams (
    dbViaHeaderId      stdViaHeaderId,
    dbViaParam**       viaParams
);
```

Description

Gets the standard via parameters associated with the specified viaHeader identified by *stdViaHeaderId*. These parameters are common to all standard vias for a specific standard via definition.

Note: The function allocates memory for the viaParams and it should be freed by using *dbFreeViaParam* when it is no longer needed.

Arguments

<i>stdViaHeaderId</i>	The database <i>dbViaHeaderId</i> for the <i>stdViaHeader</i> object.
<i>viaParams</i>	A pointer to the allocated via parameters available using the <i>dbViaParam*</i> data type.

Return Values

Boolean	Returns TRUE if standard via parameters associated with the specified viaHeader were successfully retrieved. Returns FALSE otherwise.

dbGetCustomViaHeaderParams

Boolean

```
dbGetCustomViaHeaderParams (
    dbViaHeaderId      customViaHeaderId,
    unsigned int*       numParams,
    dbParamRec**        params
);
```

Description

Gets the custom via overridden parameters associated with the specified viaHeader identified by *customViaHeaderId*. The number of custom via parameters are returned with *numParams* while the parameter array is returned with *params*. These parameters are common to all custom vias for a specific custom via definition.

Note: The function allocates memory for the parameter array and it should be freed when it is no longer needed.

Arguments

<i>customViaHeaderId</i>	The database <i>dbViaHeaderId</i> for the customViaHeader object.
<i>numParams</i>	The number of custom via parameters.
<i>params</i>	A pointer to the populated parameter array to be returned.

Return Values

Boolean	Returns TRUE if custom via parameters associated with the specified viaHeader were successfully retrieved. Returns FALSE otherwise.

dbGetCustomViaHeaderAllParams

Boolean

```
dbGetCustomViaHeaderAllParams (  
    dbViaHeaderId      customViaHeaderId,  
    unsigned int*       numParams,  
    dbParamRec**       params  
);
```

Description

Gets all the custom via parameters (overrides and default) associated with the specified viaHeader identified by *customViaHeaderId*. The number of custom via parameters are returned with numParams while the parameter array is returned with params. These parameters are common to all custom vias for a specific custom via definition.

Note: The function allocates memory for the parameter array and it should be freed when it is no longer needed.

Arguments

<i>customViaHeaderId</i>	The database dbViaHeaderId for the customViaHeader object.
<i>numParams</i>	The number of custom via parameters.
<i>params</i>	A pointer to the populated parameter array to be returned.

Return Values

Boolean	Returns TRUE if custom via parameters (overrides and default) associated with the specified viaHeader were successfully retrieved. Returns FALSE otherwise.

dbGetCustomViaHeaderLibName

Boolean

```
dbGetCustomViaHeaderLibName (  
    dbViaHeaderId      viaHeaderId,  
    String*            libName  
  
    );
```

Description

Gets the name of the master cellview library (based upon its custom via definition) associated with the specified viaHeader object in the libName parameter.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the customViaHeader object.
<i>libName</i>	A pointer to the library name to be populated.

Return Values

Boolean	Returns TRUE if the master cellview library (based upon its custom via definition) associated with the specified viaHeader object in the libName parameter was successfully retrieved. Returns FALSE otherwise.

dbGetCustomViaHeaderCellName

Boolean

```
dbGetCustomViaHeaderCellName (  
    dbViaHeaderId      viaHeaderId,  
    String*            cellName  
);
```

Description

Gets the name of the master cellview cell (based upon its custom via definition) associated with the specified viaHeader object in the cellName parameter.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the customViaHeader object.
<i>cellName</i>	A pointer to the cell name to be populated.

Return Values

Boolean	Returns TRUE if the name of the master cellview cell (based upon its custom via definition) associated with the specified viaHeader object in the cellName parameter was successfully retrieved. Returns FALSE otherwise.

dbGetCustomViaHeaderViewName

Boolean

```
dbGetCustomViaHeaderViewName (
    dbViaHeaderId      viaHeaderId,
    String*             viewName
);
```

Description

Gets the name of the master cellview view (based upon its custom via definition) associated with the specified viaHeader object in the viewName parameter.

Arguments

<i>viaHeaderId</i>	The database dbViaHeaderId for the customViaHeader object.
<i>viewName</i>	A pointer to the view name to be populated.

Return Values

Boolean	Returns TRUE if the name of the master cellview view (based upon its custom via definition) associated with the specified viaHeader object in the viewName parameter was successfully retrieved. Returns FALSE otherwise.

Site Definition Functions

This section describes functions that affect site definition objects.

About siteDefs

A site definition (siteDef) is a technology object. A siteDef defines a building element, a site for rows. A row is a repetition of a site or an array of sites. The siteDef associated with the row defines at which sites instances can be placed in the row.

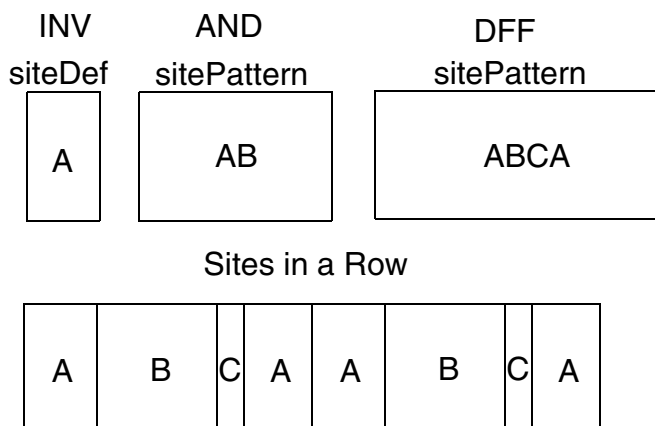
There are two predefined types for a site, `pad` and `core`. Pad sites create rows to place the chip IO pads. Core sites create rows for standard cell placement.

A siteDef can be scalar or array. An array siteDef defines an array of sites. Each site in the array is bound to a unique scalar siteDef. An array of siteDefs is also called a sitePattern.

A cellview has one sitePattern defined. The sitePattern contains a one dimensional array of sites. The sitePattern indicates the sites in the row to which the cell can snap.

In the figure below, the following rules apply.

- An instance of INV is associated with siteDef A can snap to any A site in the row.
- An instance of AND is associated with sitePattern AB can snap to any AB pattern in the row.
- An instance of DFF is associated with sitePattern ABCA can snap to any ABCA pattern in the row.



Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

If the `sitePattern` has a size of one, then it has only one site defined. The instance can snap to any site in the row that is associated with the scalar `siteDef` having the site name, for example, INV.

If the `sitePattern` has a size greater than one, then the instance snaps to a pattern in a row that is specified by the `sitePattern`, for example, AND, DFF, etc.

Two definitions support the functions.

```
typedef techId techScalarSiteDefId;  
typedef techId techArraySiteDefId;
```

The follow structure supports `siteRef`.

```
typedef struct techSiteRef techSiteRef;  
  
struct techSiteRef {  
    String      siteName;  
    dbPoint     offset;  
    dbOrient    orient;  
};
```


techIsSiteDefId

Boolean

```
techIsSiteDefId(  
    techSiteDefId    siteDef  
);
```

Description

Verifies that the given *siteDef* refers to a siteDef object.

Arguments

<i>siteDef</i>	The database techSiteDefId for the siteDef object.

Return Values

Boolean	Returns TRUE if the <i>siteDef</i> refers to a siteDef object. Returns FALSE if it is not a siteDef object.

techDeleteSiteDef

Boolean

```
techDeleteSiteDef(  
    techSiteDefId    siteDefId  
);
```

Description

Deletes the siteDef object from the specified technology file.

Arguments

<i>siteDefId</i>	The database id of the technology file siteDef object.

Return Values

Boolean	Returns TRUE if the siteDef object was successfully deleted. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techFindSiteDefByName

```
techSiteDefId
techFindSiteDefByName (
    techFileId      techId,
    String           name
);
```

Description

Searches the technology file specified by *techId* for a siteDef object with the specified name.

Arguments

<i>techId</i>	The database id of the technology file.
<i>String</i>	The name of the siteDef object to be searched for.

Return Values

<i>techSiteDefId</i>	The database id of the techSiteDef object found.

techGetSiteDefName

String

```
techGetSiteDefName(  
    techSiteDefId    siteDefId  
);
```

Description

Gets the name of the specified siteDef object.

Arguments

<i>siteDefId</i>	The database id of the technology file siteDef object.

Return Values

String	The name of the siteDef object.

techGetSiteDefType

```
techSiteDefType  
techGetSiteDefType(  
    techSiteDefId    siteDefId  
);
```

Description

Gets the type of the specified siteDef object.

Arguments

<i>siteDefId</i>	The database id of the technology file siteDef object.

Return Values

techSiteDefType	The type of the siteDef object. The choices are: pad core

techGetSiteDefWidth

```
unsigned int  
techGetSiteDefWidth(  
    techSiteDefId    siteDefId  
);
```

Description

Gets the width of the specified siteDef object.

Arguments

<i>siteDefId</i>	The database id of the technology file siteDef object.

Return Values

unsigned int	The width of the siteDef object.

techGetSiteDefHeight

```
unsigned int  
techGetSiteDefHeight(  
    techSiteDefId    siteDefId  
);
```

Description

Gets the height of the specified siteDef object.

Arguments

<i>siteDefId</i>	The database id of the technology file siteDef object.

Return Values

unsigned int	The height of the siteDef object.

techIsSiteDefSymmetricInX

Boolean

```
techIsSiteDefSymmetricInX(  
    techSiteDefId    siteDefId  
);
```

Description

Determines whether the siteDef object has the symmetric flag for the X direction set. A symmetricX flag TRUE allows the instance to be placed as R0 or MX.

Arguments

<i>siteDefId</i>	The tech file id of the siteDef object.

Return Values

Boolean	Returns TRUE if the siteDef has the symmetric flag for the X direction set. Returns FALSE otherwise.

techIsSiteDefSymmetricInY

Boolean

```
techIsSiteDefSymmetricInY(  
    techSiteDefId    siteDefId  
);
```

Description

Determines whether the siteDef object has the symmetric flag for the Y direction set. A symmetricY flag TRUE allows the instance to be placed as R0 or MY.

Arguments

<i>siteDefId</i>	The tech file id of the siteDef object.

Return Values

Boolean	Returns TRUE if the siteDef has the symmetric flag for the Y direction set. Returns FALSE otherwise.

techIsSiteDefSymmetricInR90

Boolean

```
techIsSiteDefSymmetricInR90(  
    techSiteDefId    siteDefId  
);
```

Description

Determines whether the specified siteDef object has the symmetric flag for 90 degree rotation set. A symmetricR90 flag TRUE allows the instance to be placed at a 90 degrees.

Arguments

<i>siteDefId</i>	The tech file id of the siteDef object.

Return Values

Boolean	Returns TRUE if the siteDef has the symmetric flag for 90 degree rotation set. Returns FALSE otherwise.

techSetSiteDefSymmetricInX

Boolean

```
techSetSiteDefSymmetricInX(  
    techSiteDefId    siteDefId,  
    Boolean          symmetricInX  
);
```

Description

Sets the symmetric flag for the X direction for the specified siteDef object. Setting a symmetricX flag TRUE allows the instance to be placed as R0 or MX.

Arguments

<i>siteDefId</i>	The tech file id of the siteDef object.
<i>Boolean</i>	The symmetric flag for the X direction.

Return Values

Boolean	Returns TRUE if the symmetric flag for the X direction is set successfully. Returns FALSE otherwise.

techSetSiteDefSymmetricInY

Boolean

```
techSetSiteDefSymmetricInY(  
    techSiteDefId    siteDefId  
);
```

Description

Sets the symmetric flag for the Y direction for the specified siteDef object. Setting a symmetricY flag TRUE allows the instance to be placed as R0 or MY.

Arguments

<i>siteDefId</i>	The tech file id of the siteDef object.
<i>Boolean</i>	The symmetric flag for the Y direction.

Return Values

Boolean	Returns TRUE if the symmetric flag for the Y direction is set successfully. Returns FALSE otherwise.

techSetSiteDefSymmetricInR90

Boolean

```
techSetSiteDefSymmetricInR90(  
    techSiteDefId    siteDefId  
);
```

Description

Sets the symmetric flag for 90 degree rotation for the specified siteDef object. Setting a symmetricR90 flag to TRUE allows the instance to be placed at a 90 degrees.

Arguments

<i>siteDefId</i>	The tech file id of the siteDef object.

Return Values

Boolean	Returns TRUE if the symmetric flag for 90 degree rotation is set successfully. Returns FALSE otherwise.

techSetSiteDefType

Boolean

```
techSetSiteDefType (
    techSiteDefId      siteDefId,
    techSiteDefType    sType
);
```

Description

Sets the type associated with the specified siteDef.

Arguments

<i>siteDefId</i>	The database id of the technology file siteDef object.
<i>sType</i>	The type to set the siteDef object to. The choices are: pad core

Return Values

Boolean	Returns TRUE if the type for the specified siteDef was successfully set. Returns FALSE otherwise.

techStartGenSiteDef

```
techGenStateId  
techStartGenSiteDef(  
    techFileId    techId  
);
```

Description

Starts generating all the siteDefs in the given techfile.

Arguments

<i>techId</i>	The database <code>techFileId</code> for the techfile.

Return Values

<code>techGenStateId</code>	Returns the generator <code>techGenStateId</code> used to track the state of siteDef generation.

techGenSiteDef

Boolean

```
techGenSiteDef(  
    techGenStateId    pState,  
    techSiteDefId     *pSiteDefId  
);
```

Description

Generates the next siteDef in the subject techfile.

Arguments

<i>pState</i>	The generator <code>techGenStateId</code> used to track the state of siteDef generation.
<i>*pSiteDefId</i>	The <code>techSiteDefId</code> pointer. Returned by the function if a siteDef is generated.

Return Values

Boolean	Returns TRUE if a siteDef is generated. Returns FALSE otherwise.

techGenSiteDefToRowHeader

Boolean

```
techGenSiteDefToRowHeader(  
    techGenStateId    pState,  
    dbRowHeaderId     *rowHeaderId  
);
```

Description

Generates the next rowHeader in the open cellviews.

Arguments

<i>pState</i>	The generator <code>techGenStateId</code> used to track the state of rowHeader generation.
<i>*rowHeaderId</i>	The <code>dbRowHeaderId</code> pointer. Returned by the function if a rowHeader is generated.

Return Values

Boolean	Returns TRUE if a rowHeader is generated. Returns FALSE otherwise.

techStartGenCellViewSiteDefs

```
techGenStateId  
techStartGenCellViewSiteDefs (  
    dbCellViewfId      cvfId  
);
```

Description

Starts generating all the siteDefs in the cellview.

Arguments

<i>cvId</i>	The database id of the cellview object.

Return Values

<i>techGenStateId</i>	Returns the generator <code>techGenStateId</code> used to track the state of siteDef generation.

techGenCellViewSiteDefs

Boolean

```
techGenCellViewSiteDefs(  
    dbCellViewfId      pState,  
    techSiteDefId      *pSiteDefId  
);
```

Description

Generates the next siteDef in the subject cellviews.

Arguments

<i>pState</i>	The generator used to track the state of siteDef generation.
<i>*pSiteDefId</i>	The pSiteDefId pointer. Returned by the function if a siteDef is generated.

Return Values

Boolean	Returns TRUE if a siteDef is generated. Returns FALSE otherwise.

techStringToSiteDefType

```
techSiteDefType  
techStringToSiteDefType(  
    String      s  
);
```

Description

Gets the corresponding siteDefType according to the string passed in.

Arguments

<i>s</i>	The string indicating the siteDef type.

Return Values

techSiteDefType	The siteDef type. The choices are: techcUnknownSiteDefType, techcPadSiteDefType, techcCoreSiteDefType

techSiteDefTypeToString

```
String  
techSiteDefTypeToString(  
    techSiteDefType    enumVal  
);
```

Description

Gets a string which represents the type of the siteDef.

Arguments

<i>enumVal</i>	The value of the siteDef type.

Return Values

<i>String</i>	The string representing the siteDef type. The choices are: 0 = unknown 1 = pad 2 = core

techListToSiteDefType

```
techSiteDefType  
techListToSiteDefType(  
    String      funcName,  
    list        l  
);
```

Description

Gets the corresponding siteDef type according to the list passed in.

Arguments

<i>funcName</i>	The name of the calling function.
<i>l</i>	The list passed in.

Return Values

techSiteDefType	The siteDef type. The choices are: techcUnknownSiteDefType, techcPadSiteDefType, techcCoreSiteDefType

techList2SiteDef

```
techSiteDefId  
techList2SiteDef(  
    String      funcName,  
    list        l  
);
```

Description

Gets the corresponding siteDef type according to the list passed in.

Arguments

<i>funcName</i>	The name of the calling function.
<i>l</i>	The list passed in.

Return Values

techSiteDefId	The siteDef database id.

techIsScalarSiteDefId

Boolean

```
techIsScalarSiteDefId(  
    techId      objId  
);
```

Description

Verifies that the *objId* refers to a valid oaScalarSiteDef object.

Arguments

<i>objId</i>	The <i>techId</i> for the siteDef object.

Return Values

Boolean	Returns TRUE if the <i>objId</i> refers to an oaScalarSiteDef object. Returns FALSE otherwise.

techCreateScalarSiteDef

```
techScalarSiteDefId  
techCreateScalarSiteDef(  
    techFileId      techId,  
    const char      *siteDefName,  
    techSiteDefType  type,  
    unsigned int     width,  
    unsigned int     height,  
    Boolean          isSymmetricInX,  
    Boolean          isSymmetricInY,  
    Boolean          isSymmetricInR90  
);
```

Description

Creates a scalar siteDef object using the specified attributes.

Arguments

<i>techId</i>	The <i>techId</i> for the siteDef object.
<i>*siteDefName</i>	Name of the siteDef.
<i>type</i>	Type of the siteDef, pad or core.
<i>width</i>	Width of the site defined by the siteDef.
<i>height</i>	Height of the site defined by the siteDef.
<i>isSymmetricInX</i>	Symmetric flag in the X direction of the site defined by the siteDef. The default is FALSE.
<i>isSymmetricInY</i>	Symmetric flag in the Y direction of the site defined by the siteDef. The default is FALSE.
<i>isSymmetricInR90</i>	Rotation of the site defined by the siteDef. The default is FALSE.

Return Values

<i>techScalarSiteDefId</i>	Returns the <i>techId</i> of the oaScalarSiteDef if it is created. Returns a NULL id otherwise.

techSetScalarSiteDefWidth

Boolean

```
techSetScalarSiteDefWidth(  
    techScalarSiteDefId    siteDefId,  
    unsigned int           width  
);
```

Description

Sets the width of the scalar siteDef object.

Arguments

<i>siteDefId</i>	The <code>siteDefId</code> of the siteDef object.
<i>width</i>	Width of the siteDef object.

Return Values

Boolean	Returns TRUE if the <code>siteDefId</code> width is set successfully. Returns FALSE otherwise.

techSetScalarSiteDefHeight

Boolean

```
techSetScalarSiteDefHeight(  
    techScalarSiteDefId    siteDefId,  
    unsigned int           height  
);
```

Description

Sets the height of the scalar siteDef object.

Arguments

<i>siteDefId</i>	The <code>siteDefId</code> of the siteDef object.
<i>height</i>	Height of the siteDef object.

Return Values

Boolean	Returns TRUE if the <code>siteDefId</code> height is set successfully. Returns FALSE otherwise.

techIsArraySiteDefId

Boolean

```
techIsArraySiteDefId,  
    techId      objId  
    );
```

Description

Verifies that the *objId* refers to a valid oaArraySiteDef object.

Arguments

<i>objId</i>	The <i>techId</i> for the siteDef object.

Return Values

Boolean	Returns TRUE if the <i>objId</i> refers to an oaArraySiteDef object. Returns FALSE otherwise.

techCreateArraySiteDef

```
techArraySiteDefId
techCreateArraySiteDef(
    techFileId      techId,
    const char      *siteDefName,
    techSiteDefType  type,
    unsigned int     patternSize,
    const techSiteRef *sitePattern,
    Boolean          isSymmetricInX,
    Boolean          isSymmetricInY,
    Boolean          isSymmetricInR90
);
```

Description

Creates an array siteDef object using the specified attributes. A pre-defined array of techSiteRefs defines the sitePattern of the array siteDef. The siteRef number must match the array size. The siteRef array is not checked for correctness.

The function creates the array siteDef as follows.

- creates an oaSiteRef for each techSiteRef structure.
- creates an oaSitePattern using these oaSiteRefs.
- creates an arraySiteDef using the oaSitePattern and other attributes.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<i>techId</i>	The <i>techId</i> of the <i>siteDef</i> object.
<i>*siteDefName</i>	Name of the <i>siteDef</i> .
<i>type</i>	Type of the <i>siteDef</i> , pad or core.
<i>patternSize</i>	Size of the <i>siteRef</i> array using the <i>siteDefName</i> .
<i>*sitePattern</i>	<i>SiteRef</i> array using the <i>siteDefName</i> .
<i>isSymmetricInX</i>	Symmetric flag in the X direction of the site defined by the <i>siteDef</i> . The default is FALSE.
<i>isSymmetricInY</i>	Symmetric flag in the Y direction of the site defined by the <i>siteDef</i> . The default is FALSE.
<i>isSymmetricInR90</i>	Rotation of the site defined by the <i>siteDef</i> . The default is FALSE.

Return Values

<i>techArraySiteDefId</i>	Returns the <i>techId</i> of the <i>oaArraySiteDef</i> if it is created. Returns a NULL id otherwise.

Site Pattern Functions

This section describes functions that affect site patterns.

techGetArraySiteDefSitePattern

Boolean

```
techGetArraySiteDefSitePattern(  
    techArraySiteDefId    siteDefId,  
    unsigned int          *patternSize,  
    techSiteRef           **sitePattern  
);
```

Description

Gets the sitePattern, which is the array of techSiteRefs, of the array siteDef object.

The returned sitePattern references a static variable maintained by the system. You do not need to free its memory. If you want to cache the returned sitePattern, do the following.

- use the [techCopySitePattern](#) function to copy the sitePattern.
- use the [techFreeSitePattern](#) function to free the copied sitePattern.

Arguments

<i>siteDefId</i>	The siteDef object for which the function gets the sitePattern.
<i>*patternSize</i>	Size of the techSiteRef array.
<i>**sitePattern</i>	Pointer to the static system variable that holds the techSiteRef array.

Return Values

Boolean	Returns TRUE if the function gets the sitePattern successfully. Returns FALSE otherwise.

techSetArraySiteDefSitePattern

Boolean

```
techSetArraySiteDefSitePattern(  
    techArraySiteDefId    siteDefId,  
    unsigned int          patternSize,  
    const techSiteRef     *sitePattern  
);
```

Description

Sets the sitePattern, which is the array of techSiteRefs, of the array siteDef object.

Arguments

<i>siteDefId</i>	The siteDef object for which the function gets the sitePattern.
<i>patternSize</i>	Size of the techSiteRef array.
<i>*sitePattern</i>	Pointer to the static system variable that holds the techSiteRef array.

Return Values

Boolean	Returns TRUE if the sitePattern is set successfully. Returns FALSE otherwise.

techCopySitePattern

```
techSiteRef *  
techCopySitePattern(  
    unsigned int    patternSize,  
    techSiteRef     *sitePattern  
);
```

Description

Copies the sitePattern into dynamically allocated memory. Once the copy completes, be sure to free the memory using the [techFreeSitePattern](#) function.

Arguments

<i>patternSize</i>	Size of the techSiteRef array.
<i>*sitePattern</i>	Pointer to the static system variable that holds the techSiteRef array.

Return Values

<i>techSiteRef *</i>	Pointer to the memory holding the sitePattern copy.

techFreeSitePattern

```
void  
techFreeSitePattern(  
    techSiteRef    *sitePattern  
);
```

Description

Frees the memory allocated for the sitePattern. Use after executing the [techCopySitePattern](#) function.

The function also checks whether the sitePattern is the system managed copy. The memory is freed only if the sitePattern is not the system managed copy.

Arguments

<i>*sitePattern</i>	Pointer to the memory allocated for the sitePattern.

Return Values

void	Returns a void is the memory is freed successfully.

dbGetCellViewSitePattern

Boolean

```
dbGetCellViewSitePattern(  
    dbCellViewId      cvId,  
    unsigned int       *patternSize,  
    techSiteRef        **sitePattern  
);
```

Description

Gets the sitePattern, which is the array of techSiteRefs, in the cellview.

Arguments

<i>cvId</i>	The database id of the cellview.
<i>*patternSize</i>	The number of siteRefs in the array.
<i>**sitePattern</i>	Pointer to the system variable that holds the siteRef array.

Return Values

Boolean	Returns TRUE with the number of siteRefs in the sitePattern and a pointer to the siteRef array. If the cellview does not have a sitePattern or if the sitePattern is NULL, the function returns TRUE with the number of siteRefs set to 0 and the siteRef array set to NULL.

dbSetCellViewSitePattern

Boolean

```
dbSetCellViewSitePattern(  
    dbCellViewId      cvId,  
    unsigned int      *patternSize,  
    const techSiteRef **sitePattern  
);
```

Description

Sets the sitePattern in the cellview.

Arguments

<i>cvId</i>	The database id of the cellview.
<i>*patternSize</i>	The number of siteRefs in the array. If the number of siteRefs is 0, it is equivalent to deleting the sitePattern in the cellview. OpenAccess uses this method to delete a cellview sitePattern.
<i>**sitePattern</i>	Pointer to the system variable that holds the siteRef array.

Return Values

Boolean	Returns TRUE if the sitePattern is set successfully. Returns FALSE otherwise.

Physical Constraint Functions

The Physical Rules constraint functions in this section pertain to spacing constraints.

You can view layer-purpose pairs selectively. When you create separate layer-purpose pairs for the same layer and then specify that only the layer in a layer-purpose pair to be seen, you see the virtual layer, indicated by the `techLayer` argument.

The spacing constraint functions are designed to avoid problems between spacing constraints and this selective viewing of layers. When you apply a constraint to a layer, you are implying that the constraint applies to all layer-purpose pairs referencing that layer, unless the constraint is specifically overridden for a particular layer-purpose pair.

Layer-purpose pairs are supported in DFII only. OpenAccess does not support them.

techStartGenSpacingRule

```
techRuleGenStateId  
techStartGenSpacingRule(  
    techFileId techfile  
);
```

Description

Starts the generator for generating all defined spacing constraints in the technology file. Spacing constraints can be for one or two layers.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techRuleGenStateId</code>	Returns the generator state identifier to use when generating spacing constraints.

techGenSpacingRule

Boolean

```
techGenSpacingRule(  
    techRuleGenStateId    stateId,  
    String                *ruleName,  
    techLayerNum          *pLayer1Num,  
    techPurpose           *pPurpose1,  
    techLayerNum          *pLayer2Num,  
    techPurpose           *pPurpose2,  
    techPropType          *pType,  
    techPropValue         *pValue  
);
```

Description

Generates the next spacing constraint.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

stateId	The generator state identifier.
*ruleName	The spacing constraint name to generate.
*pLayer1Num	The layer number of the first layer-purpose pair.
*pPurpose1	The purpose number of the first layer-purpose pair.
*pLayer2Num	The layer number of the second layer-purpose pair, if a two-layer constraint is being generated.
*pPurpose2	The purpose number of the second layer-purpose pair, if a two-layer constraint is being generated.
*pType	The type of the spacing constraint value.
*pValue	The spacing constraint value.

Return Value

Boolean	Returns FALSE if the generator is finished or the generator state identifier is invalid.

Example

```
techFileId techfile;
techRuleGenStateId stateId;
String ruleName, layerName;
techLayerNum layer1Num, layer2Num;
techPurpose purpose1, purpose2;
techPropType type;
techPropValue value;
stateId = techStartGenSpacingRule (techfile);
while (techGenSpacingRule (stateId, &ruleName, &layer1Num, &purpose1, &layer2Num,
&purpose2, &type, &value)) {
    switch (type) {
        case dbcIntType:
            the spacing rule value can be found value.aInt;
            break;
        case dbcFloatType:
            the spacing rule value can be found value.aFloat;
            break;
        default:
            not a numeric so not a legal spacing rule value.
    }
}
techStopRuleGen (stateId);
```

techGetSpacingRule

Boolean

```
techGetSpacingRule(  
    techFileId      techfile,  
    String           ruleName,  
    techLayerNum     layer1Num,  
    techPurpose      purpose1,  
    techLayerNum     layer2Num,  
    techPurpose      purpose2,  
    techPropType     *pType,  
    techPropValue    *pValue  
);
```

Description

Returns the value of the specified spacing constraint defined in the technology file.

The spacing constraints are located in a `spacings` construct in the technology file. They define the spacing requirements of your design environment.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The spacing constraint name for which you are looking.
<code>layer1Num</code>	The layer number of the first layer-purpose pair.
<code>purpose1</code>	The purpose number of the layer-purpose pair of layer 1.
<code>layer2Num</code>	The layer number of the layer-purpose pair of layer 2, if a two-layer constraint is being retrieved. Use <code>techcNullLayer</code> if only a one-layer constraint is being retrieved.
<code>purpose2</code>	The purpose number of the layer-purpose pair of layer 2. Use <code>techcNullLayer</code> if only a one-layer constraint is being retrieved.
<code>*pType</code>	The type of the spacing constraint value.
<code>*pValue</code>	The spacing constraint value.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the technology file does not exist or there is no spacing constraint defined for the specified layer or layers.

Example

The example for `techGetSpacingRule` is based on the values returned for the following example for [“techSetSpacingRule”](#) on page 1524:

```
techSetSpacingRule (techFile "minWidth" 0.6 "metall")
techSetSpacingRule (techFile "minWidth" 2.0 ("metall" "power"))
```

The value of 0.6 set in the `techSetSpacingRule` example is returned for any query in which the layer is "metall" or for a query in which the layer in a layer-purpose pair is equal to "metall" and purpose is *not* equal to "power".

The value of 2.0 set in the `techSetSpacingRule` example is returned if the ("metall" "power") is queried (an exact match).

```
techGetSpacingRule (techfile "minWidth" "metall")
techGetSpacingRule (techfile "minWidth" ("metall" "power"))
```

techSetSpacingRule

Boolean

```
techSetSpacingRule(  
    techFileId      techfile,  
    String          ruleName,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Updates the value of the spacing constraint for the specified layer(s) in the technology file.

The spacing constraints are located in a `spacings` construct. They define the spacing requirements of your design environment. If you do not specify a second layer, the spacing constraint applies only to the first layer.

If a spacing constraint does not exist for the specified layer(s), the function creates one.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The spacing constraint name you are looking for.
<code>layer1Num</code>	The layer number of the first layer-purpose pair.
<code>purpose1</code>	The purpose number of the first layer-purpose pair.
<code>layer2Num</code>	The layer number of the second layer-purpose pair, if a two-layer constraint is being updated. Use <code>techcNullLayer</code> if only a one-layer constraint is updated.
<code>purpose2</code>	The purpose number of the second layer-purpose pair. Use <code>techcNullLayer</code> if only a one-layer constraint is updated.
<code>type</code>	The type of the spacing constraint.
<code>value</code>	The value for the spacing constraint.

Return Value

Boolean	Returns FALSE if the technology file or layers do not exist.

Example

```
techSetSpacingRule(techFile "minWidth" 0.6 "metall")  
techSetSpacingRule(techFile "minWidth" 2.0 '("metall" "power"))
```

This example creates two separate spacing constraints "minWidth" relating to `techLayer` "metall". See the example for [“techGetSpacingRule”](#) on page 1522 for retrieving the values set in this example.

techStartGenOrderedSpacingRule

```
techGenStateId  
techStartGenOrderedSpacingRule(  
    techFileId techfile  
);
```

Description

Starts the generator for generating all defined spacing constraints in the technology file. This function is the same as `techStartGenSpacingRule`, except that this function is for spacing constraints for two layers only. The order of the layers is significant.

The following is an example of an unordered spacing constraint in which the order inside the left parentheses is equivalent to the order inside the right parentheses.

```
(minSpacing .25 layer1 layer2) == (minSpacing .25 layer2 layer1).
```

The following is an example of an ordered spacing constraint in which the order inside the left parentheses is *not* equivalent to the order inside the right parentheses:

```
(minEnclosure .35 layer1 layer2) != (minEnclosure .35 layer2 layer1)
```

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier to be used when generating spacing constraints.

techGenOrderedSpacingRule

Boolean

```
techGenOrderedSpacingRule(  
    techGenStateId    stateId,  
    String             *ruleName,  
    techLayerNum       *pLayer1Num,  
    techPurpose        *pPurpose1,  
    techLayerNum       *pLayer2Num,  
    techPurpose        *pPurpose2,  
    techPropType       *pType,  
    techPropValue      *pValue  
);
```

Description

Generates all defined spacing constraints in the technology file. The order of the layers is significant.

Arguments

stateId	The generator state identifier to be used in generating defined spacing constraints.
*ruleName	The next constraint name to be generated.
*pLayer1Num	The layer number of the layer-purpose pair of layer 1.
*pPurpose1	The purpose number of the layer-purpose pair of layer 2.
*pLayer2Num	The layer number of the layer-purpose pair of layer 2.
*pPurpose2	The purpose number of the layer-purpose pair of layer 2.
*pType	The ordered constraint type.
*pValue	The ordered constraint value.

Return Value

Boolean	Returns TRUE if all *p values returned are valid and can be used. Returns FALSE when the generator is finished.

techGetOrderedSpacingRule

Boolean

```
techGetOrderedSpacingRule(  
    techFileId      techfile,  
    String          ruleName,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techPropType    *pType,  
    techPropValue   *pValue  
);
```

Description

Returns a list of all the ordered spacing constraints defined in the `orderedSpacings` constraints of the technology file.

The `orderedSpacings` constraints define the spacing requirements for which the order of the layers is important.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The constraint for which you are looking.
<code>layer1Num</code>	The layer number of layer 1 of the layer-purpose pair.
<code>purpose1</code>	The purpose number of layer 2 of the layer-purpose pair.
<code>layer2Num</code>	The layer number of layer 2 of the layer-purpose pair.
<code>purpose2</code>	The purpose number of layer 2 of the layer-purpose pair.
<code>*pType</code>	The type of the ordered constraint.
<code>*pValue</code>	The value of the ordered constraint.

Return Value

Boolean	Returns FALSE if the technology file does not exist or there are no spacing constraints defined.

techSetOrderedSpacingRule

Boolean

```
techSetOrderedSpacingRule(  
    techFileId      techfile,  
    String          ruleName,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Updates the value of the spacing constraint for the specified layers in the `orderedSpacings` constraints of the technology file.

The `orderedSpacings` constraints define the spacing requirements for which the order of the layers is important. If an ordered spacing constraint does not exist for the specified layers, the function creates one. If an `orderedSpacings` constraints does not exist, this function creates one with the specified data.

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The constraint for which you are looking.
<code>layer1Num</code>	The layer number of the first layer-purpose pair.
<code>purpose1</code>	The purpose number of the first layer-purpose pair.
<code>layer2Num</code>	The layer number of layer 2 of the layer-purpose pair.
<code>purpose2</code>	The purpose number of layer 2 of the layer-purpose pair.
<code>type</code>	The type of the ordered spacing constraint.
<code>value</code>	The value of the ordered spacing constraint.

Return Value

Boolean	Returns FALSE if the technology file or layers do not exist.

Electrical Rule Functions

Electrical rules, are stored as layer properties in the `LayerDefinitions` section of the technology file. Electrical constraints are visible in ASCII as layer properties.

User defined electrical constraints, or rules, map to the `electricalCharacterizations` section within a constraint group.

techStartGenElectricalRule

```
techRuleGenStateId  
techStartGenElectricalRule(  
    techFileId techfile  
);
```

Description

Starts generating all electrical rules defined in the technology file. The electrical rules generated can be for one or two layers.

Arguments

<code>techfile</code>	The database identifier of the technology file where the rules are defined.

Return Value

<code>techRuleGenStateId</code>	Returns the generator state identifier to be used when generating electrical rules.

techGenElectricalRule

Boolean

```
techGenElectricalRule(  
    techRuleGenStateId    stateId,  
    String                *ruleName,  
    techLayerNum          *pLayer1Num,  
    techPurpose           *pPurpose1,  
    techLayerNum          *pLayer2Num,  
    techPurpose           *pPurpose2,  
    techPropType          *pType,  
    techPropValue         *pValue  
);
```

Description

Generates all defined electrical rules in the technology file. The electrical rules generated can be for one or two layers.

Arguments

techRuleGenStateId	The generator state identifier to be used when generating electrical rules.
*ruleName	The next rule name to be generated.
*pLayer1Num	The layer number of the layer-purpose pair of layer 1.
*pPurpose1	The purpose number of the layer-purpose pair of layer 2.
*pLayer2Num	The layer number of the layer-purpose pair of layer 2.
*pPurpose2	The purpose number of the layer-purpose pair of layer 2.
*pType	The ordered rule type.
*pValue	The ordered rule value.

Return Value

Boolean	Returns TRUE if all *p values returned via indirection are valid. Returns FALSE when the generator function is finished.

techGetElectricalRule

Boolean

```
techGetElectricalRule(  
    techFileId      techfile,  
    String           ruleName,  
    techLayerNum     layer1Num,  
    techPurpose      purpose1,  
    techLayerNum     layer2Num,  
    techPurpose      purpose2,  
    techPropType     *pType,  
    techPropValue     *pValue  
);
```

Description

Returns a list of all the electrical constraints defined as one of the following.

- `techLayerProperties` constraints in the `layerRules` section of the technology file.
- `electricalCharacterizations` constraints within a constraint group.

These constraints specify the electrical properties of the layers in your design.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techFileId</code>	The database identifier of the technology file.
<code>ruleName</code>	The electrical rule name for which you are looking.
<code>layer1Num</code>	The layer number of the layer-purpose pair of layer 1.
<code>purpose1</code>	The purpose number of the layer-purpose pair of layer 1.
<code>layer2Num</code>	The layer number of the layer-purpose pair of layer 2, if a two-layer rule is being retrieved. Use <code>techcNullLayer</code> if only a one-layer rule is being retrieved.
<code>purpose2</code>	The purpose number of the layer-purpose pair of layer 2. Use <code>techcNullLayer</code> if only a one-layer rule is being retrieved.
<code>*pType</code>	The property type of the property value.
<code>*pValue</code>	The property value.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the technology file does not exist or there are no characterization rules defined.

techSetElectricalRule

Boolean

```
techSetElectricalRule(  
    techFileId      techfile,  
    String          ruleName,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Updates the value of the electrical constraint for the specified layer or layers. The value is updated in the `electricalCharacterizations` constraints of the technology file.

The `electricalCharacterizations` constraints specify the electrical properties of the layers in your designs. If you do not specify a second layer, the constraint applies only to the first layer. If a characterization constraint does not exist for the specified layer or layers, the function creates one with the data you specify.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The electrical rule name you are looking for.
<code>layer1Num</code>	The layer number of the layer-purpose pair of layer 1.
<code>purpose1</code>	The purpose number of the layer-purpose pair of layer 1.
<code>layer2Num</code>	The layer number of the layer-purpose pair of layer 2, if a two-layer rule is being retrieved. Use <code>techcNullLayer</code> if only a one-layer rule is being retrieved.
<code>purpose2</code>	The purpose number of the layer-purpose pair of layer 2. Use <code>techcNullLayer</code> if only a one-layer rule is being retrieved.
<code>type</code>	The property type of the property value.
<code>value</code>	The property value.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the technology file or layers do not exist.

techStartGenOrderedElectricalRule

```
techGenStateId  
techStartGenOrderedElectricalRule(  
    techFileId techfile  
);
```

Description

Starts generating the electrical constraints defined in the technology file. The order of the layers is significant.

Arguments

<code>techfile</code>	The database identifier of the technology file.

Return Value

<code>techGenStateId</code>	Returns the generator state identifier used when generating the defined electrical rules.

techGenOrderedElectricalRule

Boolean

```
techGenOrderedElectricalRule(  
    techGenStateId    stateId,  
    String             *ruleName,  
    techLayerNum       *pLayer1Num,  
    techPurpose        *pPurpose1,  
    techLayerNum       *pLayer2Num,  
    techPurpose        *pPurpose2,  
    techPropType       *pType,  
    techPropValue      *pValue  
);
```

Description

Generates the electrical constraints defined in the technology file. The order of the layers is significant.

Arguments

stateId	The generator state identifier used when generating the defined electrical rules.
*ruleName	The next rule name to be generated.
*pLayer1Num	The layer number of the layer-purpose pair of layer 1.
*pPurpose1	The purpose number of the layer-purpose pair of layer 2.
*pLayer2Num	The layer number of the layer-purpose pair of layer 2.
*pPurpose2	The purpose number of the layer-purpose pair of layer 2.
*pType	The ordered rule type.
*pValue	The ordered rule value.

Return Value

Boolean	Returns TRUE if all *p values returned via indirection are valid. Returns FALSE when the generator function is finished.

techGetOrderedElectricalRule

Boolean

```
techGetOrderedElectricalRule(  
    techFileId      techfile,  
    String          ruleName,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techPropType    pType,  
    techPropValue   pValue  
);
```

Description

Returns the value of the specified ordered characterization constraint defined in the `orderedElectricalCharacterizations` constraints in the technology file. The order of the layers is significant.

The `orderedElectricalCharacterizations` constraints specify electrical properties in which the order of layers is important.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The electrical rule name for which you are looking.
<code>layer1Num</code>	The layer number of the layer-purpose pair of layer 1.
<code>purpose1</code>	The purpose number of the layer-purpose pair of layer 1.
<code>layer2Num</code>	The layer number of the layer-purpose pair of layer 2, if a two-layer rule is being retrieved. Use <code>techcNullLayer</code> if only a one-layer rule is being retrieved.
<code>purpose2</code>	The purpose number of the layer-purpose pair of layer 2. Use <code>techcNullLayer</code> if only a one-layer rule is being retrieved.
<code>*pType</code>	The property type of the property value.
<code>*pValue</code>	The property value.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the technology file does not exist or there is no characterization constraint defined for the specified layers.

techSetOrderedElectricalRule

Boolean

```
techSetOrderedElectricalRule(  
    techFileId      techfile,  
    String          ruleName,  
    techLayerNum    layer1Num,  
    techPurpose     purpose1,  
    techLayerNum    layer2Num,  
    techPurpose     purpose2,  
    techPropType    type,  
    techPropValue   value  
);
```

Description

Updates the value of the electrical rule for the specified layers in the `orderedElectricalCharacterizations` constraints of the technology file. The order of the layers is significant.

The `orderedElectricalCharacterizations` constraints specify the electrical characteristics in which the order of layers is important. If a constraint does not exist for the specified layer or layers, the function creates one with the specified data.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Arguments

<code>techfile</code>	The database identifier of the technology file.
<code>ruleName</code>	The electrical rule name for which you are looking.
<code>layer1Num</code>	The layer number of the layer-purpose pair of layer 1.
<code>purpose1</code>	The purpose number of the layer-purpose pair of layer 1.
<code>layer2Num</code>	The layer number of the layer-purpose pair of layer 2, if a two-layer rule is being updated. Use <code>techcNullLayer</code> if only a one-layer rule is being updated.
<code>purpose2</code>	The purpose number of the layer-purpose pair of layer 2. Use <code>techcNullLayer</code> if only a one-layer rule is being updated.
<code>type</code>	The property type of the property value.
<code>value</code>	The property value.

Return Value

<code>Boolean</code>	Returns <code>FALSE</code> if the technology file or layers do not exist.

Technology File Database Functions

techGetDBUPerUU

```
double  
techGetDBUPerUU(  
    techFileId      techId,  
    dbCellViewType  cvType  
);
```

Description

Gets the number of database units per each user unit for the specified cellview type in the specified technology file.

Arguments

<i>techId</i>	The database identifier of the technology file.
<i>cvType</i>	The enum value of the cellview type.

Return Values

double	The number of database units per user unit.

techGetUserUnit

```
dbUserUnitType  
techGetUserUnit(  
    techFileId      techId,  
    dbCellViewType  cvType  
);
```

Description

Gets the user unit that is set for the specified cellview type in the specified technology file.

Arguments

<i>techId</i>	The database identifier of the technology file.
<i>cvType</i>	The enum value of the cellview type.

Return Values

dbUserUnitType	The user unit that is set for the specified cellview type in the specified technology file.

techIsDBUPerUUSet

Boolean

```
techIsDBUPerUUSet (
    techFileId      techId,
    dbCellViewType  cvType
)
```

Description

Checks whether or not the database units per user unit (DBUPerUU) value is set for the specified cellview type in the specified technology database.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>cvType</code>	The enum value of the cellview type.

Return Values

Boolean	Returns TRUE if the database units per user unit value is set for the specified cellview type in the specified technology database. Returns FALSE otherwise.

techIsUserUnitSet

Boolean

```
techIsUserUnitSet(  
    techFileId      techId,  
    dbCellViewType  cvType  
)
```

Description

Checks whether or not the user unit is set for the specified cellview type in the specified technology database.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>cvType</code>	The enum value of the cellview type.

Return Values

Boolean	Returns TRUE if the user unit is set for the specified cellview type in the specified technology database. Returns FALSE otherwise.

techSetDBUPerUU

Boolean

```
techSetDBUPerUU(  
    techFileId      techId,  
    dbCellViewType  cvType,  
    double           dbuPerUU  
);
```

Description

Set the number of database units per each user unit for the specified cellview type in the specified technology file.

Arguments

<i>techId</i>	The database identifier of the technology file.
<i>cvType</i>	The enum value of the cellview type.
<i>dbuPerUU</i>	The number of database units per user unit.

Return Values

Boolean	Returns TRUE if the number of database units per each user unit for the specified cellview type in the specified technology file is successfully set. Returns FALSE otherwise.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

techSetUserUnit

Boolean

```
techSetUserUnit(  
    techFileId      techId,  
    dbCellViewType  cvType,  
    dbUserUnitType  userUnit  
);
```

Description

Sets the user unit for the specified cellview type in the specified technology database.

Arguments

<i>techId</i>	The identifier of the technology database.
<i>cvType</i>	The enum value of the cellview type.
<i>userUnit</i>	The enum value of the user unit. Valid Values: dbcMicro dbcMillimeter dbcCentimeter dbcMeter dbcMil dbcInch

Return Values

Boolean	Returns TRUE if the user unit for the specified cellview type in the specified technology database is successfully set. Returns FALSE otherwise.

techUnsetDBUPerUU

Boolean

```
techUnsetDBUPerUU(  
    techFileId      techId,  
    dbCellViewType  cvType  
);
```

Description

Unsets the number of database units per each user unit for the specified cellview type in the specified technology database. Issues a warning if the number of database units per user unit is not set for the cellview type in the technology database.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>cvType</code>	The enum value of the cellview type.

Return Values

Boolean	Returns TRUE if the number of database units per each user unit for the specified cellview type in the specified technology database is successfully unset. Returns FALSE otherwise.

techUnsetUserUnit

Boolean

```
techUnsetUserUnit(  
    techFileId      techId,  
    dbCellViewType  cvType  
);
```

Description

Unsets the user unit for the specified cellview type in the specified technology database. Issues a warning if the user unit is not set for the cellview type in the technology database.

Arguments

<code>techId</code>	The identifier of the technology database.
<code>cvType</code>	The enum value of the cellview type.

Return Values

Boolean	Returns TRUE if the user unit for the specified cellview type in the specified technology database is successfully unset. Returns FALSE otherwise.

Trim Layer Functions

The trim layer C functions retrieve information about the trim layers and the metal and poly layers trimmed by these trim layers.

This chapter describes the following C functions:

- [techGetTrimLayer](#)
- [techGetTrimmedLayers](#)
- [techFreeColoredLayerArray](#)
- [dbGetShapeTrimFillType](#)
- [dbStartGenCellViewTrimFillShapes](#)
- [dbGenCellViewTrimFillShapes](#)

The trim layer C functions use the following enums and structure:

Enumerated constants in type **techLayerParamMaskColorType**

```
typedef enum techLayerParamMaskColorType {  
    techcLayerParamMaskAnyColor = -1,  
    techcLayerParamMaskGrayColor = 0,  
    techcLayerParamMask1Color = 1,  
    techcLayerParamMask2Color = 2,  
    techcLayerParamMask3Color = 3  
} techLayerParamMaskColorType;
```

Enumerated constants in type **techLayerParamColorLockedType**

```
typedef enum techLayerParamColorLockedType {  
    techcLayerParamAnyColorLocked = 0,  
    techcLayerParamColorLocked = 1,  
    techcLayerParamColorUnlocked = 2  
} techLayerParamColorLockedType;
```

Structure of **techColoredLayer**

```
struct techColoredLayer {  
    techLayerId                layerId;  
    techLayerParamMaskColorType color;  
    techLayerParamColorLockedType colorState;  
} techColoredLayer;
```

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Here, `layerId` is the tech layer ID, `color` is the mask color, and `colorState` depicts the state of the color.

techGetTrimLayer

Boolean

```
techGetTrimLayer(  
    techLayerId layerId,  
    techLayerParamMaskColorType color,  
    techLayerParamColorLockedType lock,  
    techColoredLayer *trimLayer  
);
```

Description

Returns the trim layer that trims the specified metal or poly layer with the given color and color state combination.

Arguments

<i>layerId</i>	Tech layer ID for metal or poly layer.
<i>color</i>	The mask color. Valid Values: techcLayerParamMaskGrayColor techcLayerParamMask1Color techcLayerParamMask2Color techcLayerParamMask3Color
<i>lock</i>	The color state. Valid Values: techcLayerParamColorLocked techcLayerParamColorUnlocked
* <i>trimLayer</i>	Output argument. Pointer to the trim layer that trims the given metal or poly layer.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Return Value

Boolean TRUE, if the operation succeeded.
 FALSE, if the operation failed.

Examples

```
techLayerNum layerNum = techGetLayerNum(techfile, "Metal1");  
techLayerId metalLayer = techGetTechLayerIdByNumber(techfile, layerNum);  
techLayerParamMaskColorType color = techcLayerParamMask1Color;  
techLayerParamColorLockedType colorState = techcLayerParamColorLocked;  
techColoredLayer trimLayer;
```

```
techGetTrimLayer(metalLayer, color, colorState, &trimLayer);
```

On success, returns trim layer along with its color attributes (color mask and color state), trimming metal layer `Metal1` with color `mask1` and color state `locked`.

Note: If no trim layer is defined for trimming metal layer `Metal1` with color `mask1` and color state `locked`, `trimLayer.layerId` is set to `techcNullLayerId`.

techGetTrimmedLayers

Boolean

```
techGetTrimmedLayers(  
    techLayerId layerId,  
    techLayerParamMaskColorType color,  
    techLayerParamColorLockedType lock,  
    unsigned int *nTrimmedLayers,  
    techColoredLayer **trimmedLayers  
);
```

Description

Returns an array of metal or poly layers trimmed by the specified layer with the given color and color state combination.

Note: Use the [techFreeColoredLayerArray](#) function to free memory allocated to the array.

Arguments

<i>layerId</i>	Tech layer ID for trim layer.
<i>color</i>	The mask color. Valid Values: techcLayerParamMaskGrayColor techcLayerParamMask1Color techcLayerParamMask2Color techcLayerParamMask3Color
<i>lock</i>	The color state. Valid Values: techcLayerParamColorLocked techcLayerParamColorUnlocked
* <i>nTrimmedLayers</i>	Output argument. Returns the number of colored metal or poly layers trimmed by the specified trim layer.
** <i>trimmedLayers</i>	Output argument. Returns an array of colored metal or poly layers trimmed by the specified trim layer.

Cadence Integrators Toolkit Database Reference

CDBA Technology File Functions

Return Value

Boolean TRUE, if the function runs successfully.
 FALSE, in case of failure.

Examples

```
techLayerNum layerNum = techGetLayerNum(techfile, "Trim1");
techLayerId trimLayer = techGetTechLayerIdByNumber(techfile, layerNum);
techLayerParamMaskColorType color = techcLayerParamMask2Color;
techLayerParamColorLockedType colorState = techcLayerParamColorUnlocked;
unsigned int *nTrimmedLayers;
techColoredLayer *trimmedLayers;
techGetTrimmedLayers(trimLayer, color, colorState, &nTrimmedLayers,
&trimmedLayers);
```

Returns an array of metal or poly layers along with their color attributes (color mask and color state), trimmed by trim layer Trim1 with color mask2, and color state unlocked.

Note: If no metal layer is trimmed by Trim1 with color mask2 and color state unlocked, nTrimmedLayers is set to 0.

```
// Free the allocated memory
techFreeColoredLayerArray(nTrimmedLayers, trimmedLayers)
```

techFreeColoredLayerArray

```
void  
techFreeColoredLayerArray(  
    unsigned int      nColoredLayers,  
    techColoredLayer  *coloredLayers  
);
```

Description

Deallocates the memory allocated to the techColoredLayer array by [techGetTrimmedLayers](#).

Arguments

<i>nColoredLayers</i>	Number of entries in the techColoredLayer array.
<i>*coloredLayers</i>	Pointer to the techColoredLayer array, for which memory is to be deallocated.

Return Value

No return value.

Example

```
techFreeColoredLayerArray(nTrimmedLayers, trimmedLayers)
```

Frees memory allocated for the techColoredLayer array.

dbGetShapeTrimFillType

```
const char* trimFillType  
dbGetShapeTrimFillType(  
    dbShapeId shapeId  
);
```

Description

Returns the type of trimFillShape for the specified trim shape.

Arguments

<i>shapeId</i>	Database ID of a shape.

Value Returned

<i>trimFillShapeType</i>	Fill type for the trimFillShape . Valid values: notTrim, bridgeMetal, SADPFillPatch, gapFill

dbStartGenCellViewTrimFillShapes

```
dbGenStateId  
dbStartGenCellViewTrimFillShapes (  
    dbCellViewId    cvId  
)
```

Description

Initializes a generator for all trimFillShape objects in the specified cellview. A generator state is returned, which is used with the [dbGenCellViewTrimFillShapes](#) function.

Arguments

<i>cvId</i>	ID of the cellview containing the trimFillShape to be generated.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

dbGenCellViewTrimFillShapes

```
Boolean  
    dbGenCellViewTrimFillShapes(  
        dbGenStateId      pState,  
        dbShapeId         *trimFillShape  
    );
```

Description

Retrieves the next trimFillShape ID from the generator with the given dbGenStateId. The trimFillShape ID is returned in the dbShapeId argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>trimFillShapeID</i>	ID of the next <i>trimFillShape</i> produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE, if the next trimFillShape ID is successfully generated. FALSE, if the list is exhausted or an error occurs.

CDBA Display Resource File Functions

This chapter describes the C functions that use the rules governing the display of design data.

A display resource file (usually named `display.drf`) contains display information which defines the colors, stipple patterns, line styles, and packets for display devices. A display device is anything that is drawn to or drawn on. It can be a computer monitor or a plotter. The default display device name is `display`, which is the computer monitor (be it color or black and white).

For more information on display resource files see the *[Virtuoso Technology Data User Guide](#)*.

For more information on display resource file functions in this chapter, see the following sections:

- [Moving the Display Resource Definition](#) on page 1562
- [Using DRM Software](#) on page 1562
- [Displays](#) on page 1563
- [Display-Dependent Attributes](#) on page 1573
- [Colors](#) on page 1601
- [Stipple Patterns](#) on page 1613
- [Line Styles](#) on page 1624
- [Packets](#) on page 1632

Moving the Display Resource Definition

You can move the display resource definition from your technology file to the display resources file so that the display resources can be shared by all layer-purpose pairs (via packet name references) in all the technology files. This serves the larger purpose of

- Simplifying your technology file
- Saving the X resources that specify color
- Supporting a non-DFII application graphic
- Sharing a display resource definition among libraries within a project

Using DRM Software

DRM (display resource management) software centralizes the management of display resources. When an application such as Layout Plus is initialized, the Display Resource Manager (DRM) loads the `display.drf` files.

Any time any of the defined resources are changed, by using the Display Resource Editor (DRE) or by calling a `drSetPacket` function, a trigger fires and informs internal applications of the change. For example, it might inform the Graphics Editor to redraw the screen.

With the new 5.00 technology file and DRM software, you can

- Specify a packet's display information
- Map a layer-purpose pair to a packet that defines the fill color, outline color, stipple pattern, or line style.

Displays

The following functions create displays, delete displays, or give information about a display.

drCreateDisplay

```
drDisplayId
drCreateDisplay(
    String          displayName,
    unsigned int    maxNumberOfColors,
    unsigned int    maxNumberOfStipples,
    unsigned int    maxNumberOfLineStyles
);
```

Description

Creates a new display whose name is `displayName` and returns the identifier of the new display. The parameters `maxNumberOfColors`, `maxNumberOfStipples`, and `maxNumberOfLineStyles` declare the maximum number of colors, stipple patterns, and line styles for the display. It returns NULL if the display already exists or if not successful.

Arguments

<code>displayName</code>	The name of the new display.
<code>maxNumberOfColors</code>	The maximum number of colors specified.
<code>maxNumberOfStipples</code>	The maximum number of stipples specified.
<code>maxNumberOfLineStyles</code>	The maximum number of line styles specified.

Return Value

<code>drDisplayId</code>	The identifier of the new display created.

drDeleteDisplay

Boolean

```
drDeleteDisplay(  
    drDisplayId  displayId  
);
```

Description

Deletes the display identified by `displayId`.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

Boolean	The function returns TRUE if the display is deleted successfully; otherwise, it returns FALSE.

drGetDisplayId

```
drDisplayId  
drGetDisplayId(  
    displayName  
);  
String  displayName;
```

Description

Returns the display identifier of the display whose name is `displayName`.

Arguments

<code>displayName</code>	The identifier of the display.

Return Value

<code>drDisplayId</code>	The function returns NULL if the display does not exist or if is not successful.

drIsDisplayId

Boolean

```
drIsDisplayId(  
    displayId  
);  
drDisplayId  displayId
```

Description

Checks whether the identifier of a display is valid.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

Boolean	Returns TRUE if <code>displayId</code> is a valid display identifier; otherwise, it returns FALSE.

drGetDisplayName

```
String  
drGetDisplayName(  
    displayId  
);  
drDisplayId  displayId;
```

Description

Returns the name of the display specified by `displayId`.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>String</code>	The name of the display. The function returns NULL if the identifier is not a valid display identifier.

drGetDisplayMaxColors

```
unsigned int  
drGetDisplayMaxColors(  
    drDisplayId    displayId  
);
```

Description

Returns the maximum number of colors allowed in the display specified by `displayId`.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>unsigned int</code>	The maximum number of colors allowed in the display. The function returns 0 if the identifier is not a valid display identifier.

drGetDisplayMaxStipples

```
unsigned int  
drGetDisplayMaxStipples(  
    displayId  
);  
drDisplayId  displayId;
```

Description

Returns the maximum number of stipple patterns allowed in the display specified by displayId.

Arguments

displayId	The identifier of the display.

Return Value

unsigned int	The maximum number of stipple patterns allowed in the display. The function returns 0 if the identifier is not a valid display identifier.

drGetDisplayMaxLineStyles

```
unsigned int  
drGetDisplayMaxLineStyles(  
    drDisplayId    displayId  
);
```

Description

Returns the maximum number of line styles allowed in the display specified by `displayId`.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>unsigned int</code>	The maximum number of line styles allowed in the display. The function returns 0 if the identifier is not a valid display identifier.

drStartGenDisplay

```
drGenStateId  
drStartGenDisplay(  
    );
```

Description

Starts the generator for all displays.

Arguments

None.

Return Value

<code>drGenStateId</code>	The generator state identifier to be used with the <code>drGenDisplay</code> function.

drGenDisplay

Boolean

```
drGenDisplay(  
    drGenStateId    state,  
    drDisplayId     *pDisplayId  
);
```

Description

This generator function generates display identifiers for all the displays.

Arguments

state	The generator state identifier returned by the drStartGenDisplay function.
*pDisplayId	The pointer to the display identifier variable.

Return Value

Boolean	If the function returns TRUE, *pDisplayId has been assigned the next valid display identifier generated. The function returns FALSE if the generator state identifier is invalid, if the display identifiers are invalid, or if the generator is finished.

Display-Dependent Attributes

Display-dependent attributes are those that refer to entries in the display's tables. When setting or retrieving these attributes, you must specify the display identifier, because it is possible that a packet may refer to two different display devices for the same attribute.

The following are functions for display-dependent attributes.

drGetPacketFillColor

Boolean

```
drGetPacketFillColor(  
    drDisplayId    displayId,  
    String          packetName,  
    drColor         **ppColor  
);
```

Description

Returns the packet fill color used in a display.

Arguments

displayId	The identifier of the display.
packetName	The name of the packet.
**ppColor	The packet fill color.

Return Value

Boolean	The function returns TRUE and the packet fill color argument updated for the specified display identifier and packet name. It returns FALSE if it is not successful.

drGetPacketFillColorIndex

```
int  
drGetPacketFillColorIndex(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet fill color specified by `displayId` and `packetName`. The color is returned as an index into the display's color table. The `drGetColor` function retrieves the RGB values given in the color index.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

<code>int</code>	The packet fill color index specified by <code>displayId</code> and <code>packetName</code> . The function returns -1 if the packet's fill color is not defined or successful.

drGetPacketFillColorName

String

```
drGetPacketFillColorName (
    drDisplayId    displayId,
    String         packetName
);
```

Description

Returns the packet's fill color name specified by `displayId` and `packetName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

String	The packet's fill color name specified by <code>displayId</code> and <code>packetName</code> . The function returns NULL if the packet's fill color is not defined or if not successful.

drSetPacketFillColor

Boolean

```
drSetPacketFillColor(  
    drDisplayId    displayId,  
    String          packetName,  
    unsigned int    colorIndex  
);
```

Description

Sets the fill color of the packet specified by `displayId` and `packetName` to the one specified by `colorIndex`. The `colorIndex` must not exceed the maximum number of entries allocated for the color table in the `drCreateDisplay` call.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>colorIndex</code>	The color index.

Return Value

Boolean	The function returns TRUE if modification is successful; otherwise, it returns FALSE.

drGetPacketOutlineColor

Boolean

```
drGetPacketOutlineColor(  
    drDisplayId    displayId,  
    String         packetName,  
    drColor        **ppColor  
);
```

Description

Returns the packet outline color specified by `displayId` and `packetName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>**ppColor</code>	The packet outline color.

Return Value

Boolean	The function returns TRUE and the packet outline color specified by <code>displayId</code> and <code>packetName</code> . It returns FALSE if not successful.

drGetPacketOutlineColorIndex

```
int  
drGetPacketOutlineColorIndex(  
    drDisplayId    displayId,  
    String          packetName  
);
```

Description

Returns the packet outline color index specified by `displayId` and `packetName`. The color is returned as an index into the display's color table. The `drGetColor` function retrieves the RGB values given in the color index.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

<code>int</code>	The packet outline color specified by <code>displayId</code> and <code>packetName</code> . The function returns -1 if the packet's outline color is not defined or if not successful.

drGetPacketOutlineColorName

String

```
drGetPacketOutlineColorName(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet's outline color name specified by `displayId` and `packetName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

String	The packet's outline color name specified by <code>displayId</code> and <code>packetName</code> . The function returns NULL if the packet's outline color is not defined or if not successful.

drSetPacketOutlineColor

Boolean

```
drSetPacketOutlineColor(  
    drDisplayId    displayId,  
    String          packetName,  
    unsigned int    colorIndex  
);
```

Description

Sets the outline color of the packet specified by `displayId` and `packetName` to the one specified by `colorIndex`. The `colorIndex` must not exceed the maximum number of entries allocated for the color table in the `drCreateDisplay` call.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>colorIndex</code>	The color index specifying the outline color.

Return Value

Boolean	The function returns TRUE if modification is successful; otherwise, it returns FALSE.

drGetPacketStipple

Boolean

```
drGetPacketStipple(  
    drDisplayId    displayId,  
    String          packetName,  
    drStipple      **ppStipple  
);
```

Description

The function returns TRUE with a NULL stipple pointer if the stipple of the specified packet is `_NA_`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>**ppStipple</code>	The packet stipple.

Return Value

Boolean	The function returns FALSE if it is not successful. The function returns TRUE and the packet stipple specified by <code>displayId</code> , and the packet name if the stipple of the specified packet is defined.

drGetPacketStippleIndex

```
int  
drGetPacketStippleIndex(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet stipple pattern index specified by `displayId` and `packetName`. The stipple pattern is returned as an index into the display's stipple table. The `drGetStipple` function retrieves the stipple pattern given in the stipple index.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

<code>int</code>	The packet stipple pattern specified by <code>displayId</code> and <code>packetName</code> . The function returns -1 if the packet's stipple pattern is not defined or if not successful.

drGetPacketStippleName

String

```
drGetPacketStippleName(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet's stipple name specified by `displayId` and `packetName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

String	The function returns NULL if the packet's stipple pattern is not defined or if not successful.

drGetPacketFillStyle

Boolean

```
drGetPacketFillStyle(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet's fill style in a display.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

Boolean	The function returns NULL if the packet's fill style is not defined or if the function is not successful.

drSetPacketFillStyle

Boolean

```
drSetPacketFillStyle(  
    drDisplayId      displayId,  
    String            packetName,  
    drPacketFillStyle fillStyle  
);
```

Description

Returns the packet's fill style in a display.

Arguments

displayId	The identifier of the display.
packetName	The name of the packet.
fillStyle	The packet's fill style.

Return Value

Boolean	The function returns NULL if the packet's fill style is not defined or if the function is not successful.

drSetPacketStipple

Boolean

```
drSetPacketStipple(  
    drDisplayId    displayId,  
    String          packetName,  
    unsigned int    stippleIndex  
);
```

Description

Sets the stipple pattern of the packet specified by `displayId` and `packetName` to the one specified by `stippleIndex`. The `stippleIndex` must not exceed the maximum number of entries allocated for the stipple table in the `drCreateDisplay` call.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>stippleIndex</code>	The stipple index.

Return Value

Boolean	The function returns TRUE if modification is successful; otherwise, it returns FALSE.

drGetPacketLineStyle

Boolean

```
drGetPacketLineStyle(  
    drDisplayId    displayId,  
    String         packetName,  
    drLineStyle    **ppLineStyle  
);
```

Description

Returns the packet line style specified by `displayId` and `packetName`. The `drGetLineStyle` function retrieves the line style pattern when given the line style index.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>**ppLineStyle</code>	The packet's line style.

Return Value

Boolean	The function returns TRUE and the packet line style specified by <code>displayId</code> and <code>packetName</code> . The function returns FALSE if the packet's line style is not defined or if not successful.

drGetPacketLineStyleIndex

```
int  
drGetPacketLineStyleIndex(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet line style index specified by `displayId` and packet name.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

<code>int</code>	Returns the index of the line style. The function returns -1 if the packet line style index is not found or if the function is not successful.

drGetPacketLineStyleName

String

```
drGetPacketLineStyleName(  
    drDisplayId    displayId,  
    String         packetName  
);
```

Description

Returns the packet's line style name specified by `displayId` and `packetName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.

Return Value

String	The function returns NULL if the packet's line style is not defined or if the function is not successful.

drSetPacketLineStyle

Boolean

```
drSetPacketLineStyle(  
    drDisplayId    displayId,  
    String          packetName,  
    unsigned int    lineStyleIndex  
);
```

Description

Sets the line style of the packet specified by `displayId` and `packetName` to the one specified by `lineStyleIndex`. The `lineStyleIndex` must not exceed the maximum number of entries allocated for the line style table in the `drCreateDisplay` call.

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>lineStyleIndex</code>	The index of the line style.

Return Value

Boolean	The function returns TRUE if modification is successful; otherwise, it returns FALSE.

drSetStipple

Boolean

```
drSetStipple(  
    drDisplayId    displayId,  
    String         stippleName,  
    unsigned int   width,  
    unsigned int   height,  
    unsigned char  *pattern  
);
```

Description

Stores stipple information in the stipple table of the display resource manager and calls the application stipple modification trigger to update the stipple's pixel map in the graphics library.

Arguments

displayId	The identifier of the display.
stippleName	The stipple name.
width	The width of the stipple.
height	The height of the stipple.
*pattern	The stipple pattern.

Return Value

Boolean	The function returns TRUE if the stipple's pixel map in the graphics library is successfully updated. Otherwise, it returns FALSE.

drFreeStipple

Boolean

```
drFreeStipple(  
    unsigned int    drDisplayId  displayId,  
    String          stippleName  
);
```

Description

Deletes the specified stipple from the DRM.

Arguments

displayId	The identifier of the display.
stippleName	The stipple name.

Return Value

Boolean	Returns TRUE if the deletion is successfully performed; otherwise, the function returns FALSE.

drSetLineStyle

Boolean

```
drSetLineStyle(  
    drDisplayId      displayId,  
    String            lineStyleName,  
    unsigned int      width,  
    unsigned int      length,  
    unsigned char     *pattern  
);
```

Description

Stores line style information in the line style table of the DRM and calls the application's line style modification trigger to update the line style's dash list in the graphics library.

Arguments

displayId	The identifier of the display.
lineStyleName	The name of the line style.
width	The width of the line.
length	The length of the line.
*pattern	The address of the line style table, returned by indirection.

Return Value

Boolean	Returns TRUE if the information is successfully stored; otherwise, the function returns FALSE.

drFreeLineStyle

Boolean

```
drFreeLineStyle(  
    drDisplayId    displayId,  
    String         lineStyleName  
);
```

Description

Deletes the specified line style from the DRM.

Arguments

displayId	The identifier of the display.
lineStyleName	The name of the line style.

Return Value

Boolean	Returns TRUE if the line style is successfully deleted. Otherwise, the function returns FALSE.

Cadence Integrators Toolkit Database Reference

CDBA Display Resource File Functions

drSetPacket

```
extern Boolean
drSetPacket (
    drDisplayId      displayId,
    String            packetName,
    String            stippleName,
    String            lineStyleName,
    String            fillColorName,
    String            outlineColorName,
    drPacketFillStyle fillStyle
);

enum drPacketFillStyle {
    drcPacketUnknownFill      = 0, /* alias layerFillUnknown */
    drcPacketOutlineFill      = 1, /* alias layerOutlineNoFill */
    drcPacketSolidFill        = 2, /* alias layerSolidFill */
    drcPacketXFill            = 3, /* alias xFill */
    drcPacketStippleFill      = 4, /* alias layerStippleFill */
    drcPacketOutlineStippleFill = 5, /* alias layerOutlineStippleFill */
    drcPacketFillStyleNotDecided = 6 /* Not decided yet. */
};
```

Description

Selectively changes items in a packet. Fill colors, outline colors, stipples, line styles, or fill styles may be changed. Items passed as NULL will not be changed. This routine does not only change these values in the DRM (design rule manager) database, but also triggers the applications to update their pen in the graphics library database. If an attribute is not NULL then that attribute updated, for example, if `stippleName == "dots"`, then the packet now uses "dots" as its stipple. The last is a `drPacketFillType` enum. If its value is set to `drPacketUnknownFill`, then no change is made, else it should be set.

Cadence Integrators Toolkit Database Reference

CDBA Display Resource File Functions

Arguments

<code>displayId</code>	The identifier of the display.
<code>packetName</code>	The name of the packet.
<code>stippleName</code>	The stipple name.
<code>lineStyleName</code>	The name of the line style.
<code>fillColorName</code>	The name of the fill color.
<code>outlineColorName</code>	The name of the outline color.
<code>fillStyle</code>	The name of the fill style.

Return Value

Boolean	The function returns TRUE if the selected items are successfully changed. Otherwise, the function returns FALSE.

drmInit

String

```
drmInit(  
    int      *pArgc,  
    char     **pArgv  
);
```

Description

Initializes the display resource manager program.

Arguments

The argument count and argument vector, as passed to your program's main function.

Return Value

String

The current version of the display resource manager software (such as 4.4.3 or 5.0.0).

Cadence Integrators Toolkit Database Reference

CDBA Display Resource File Functions

drOpenDrf

Boolean

```
drOpenDrf(  
    String      drfPath,  
    Boolean     askToSave  
);
```

Description

Opens a display resource file and gives you the option of not seeing the prompt to save upon exit. (Many users merge in or load in display resource files while editing but have no intention of saving them.)

Arguments

drfPath	The path to the display resource file.
askToSave	The option to be queried to save the changes to the display resource file.

Return Value

Boolean	The function returns TRUE if it successfully locates the path to the display resource file and loads the file. If TRUE, if you have modified but not saved any display resource file changes on exit, you are queried to save the changes. Examples of changes are using the DRE (display resource environment) or merging or loading your file into another display resource file. If the function returns FALSE, either the path to the display was not successfully located, or if it does open the display resource file, you are not prompted to save the file upon exit even if you have modified but not saved internal display resource file information.

Cadence Integrators Toolkit Database Reference

CDBA Display Resource File Functions

drSave

```
extern Boolean  
drSave(  
    String      filename  
    Boolean     incSave  
);
```

Description

Writes (saves) the updated virtual memory version of the information you have changed to a disk display resource (`display.drf`) file.

Arguments

<code>filename</code>	The name of the display resource file to save.
<code>incSave</code>	If the value of the 2nd parameter, <code>incSave</code> , is <code>TRUE</code> , <code>drm</code> saves all changed display information only. If the value of the 2nd parameter, <code>incSave</code> , is <code>FALSE</code> , <code>drm</code> saves all current display information into the specified <code>display.drf</code> file.

Return Value

<code>Boolean</code>	The function returns <code>TRUE</code> if the file is successfully saved; otherwise, it returns <code>FALSE</code> .

Colors

A packet may have three colors defined for drawing objects:

- Fill color
- Outline color
- Bright color

Unselected objects on the packet are filled with the fill color and their outline is drawn with the outline color. In most cases the fill color and the outline color will be the same.

Selected objects on the packet are drawn with bright color.

Colors are stored in a display's color table and are addressed either by the color table index or the color name. The color table index is guaranteed to be unique and to remain the same during the lifetime of the color table.

General Data Structures to Generate Colors

The following data structures are used to set up colors for objects in a packet.

drColorStructure

The color parameters are specified in the structure of type `drColor`.

```
typedef struct drColor drColor;  
struct drColor {  
    String name; unsigned long index; unsigned short red, green, blue;  
};
```

Description

Creates an array of color parameters.

Argument

<code>name</code>	The field <code>name</code> specifies the color name.
<code>index</code>	The field <code>index</code> specifies the color index in the color table.
<code>red, green, blue</code>	The fields <code>red</code> , <code>green</code> , and <code>blue</code> specify the red, green, and blue components of the color. When making a color, all the fields must be initialized.

drMakeColor

Boolean

```
drMakeColor(  
    drDisplayId displayId,  
    drColor *pColor  
);
```

Description

Makes a color in the display's color table. If the color does not exist, it is created. If the color exists, its parameters are updated. The color index uniquely identifies the color.

Arguments

<code>displayId</code>	The identifier of the display.
<code>*pColor</code>	The pointer to a <code>drColor</code> data structure.

Return Value

Boolean	The function returns TRUE if the creation or modification is successful; otherwise, it returns FALSE.

drGetNumberOfColors

```
int  
drGetNumberOfColors(  
    drDisplayId displayId  
);
```

Description

Returns the number of colors in a display specified by `displayId`.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>int</code>	The number of colors in the specified display.

drSetColorBlink

Boolean

```
drSetColorBlink(  
    drDisplayId    displayId,  
    unsigned int   colorIndex,  
    Boolean        *pBlink  
);
```

Description

Sets the specified color to blink.

Arguments

displayId	The identifier of the display.
colorIndex	The color index which uniquely identifies the color.
*pBlink	The state indicating whether the color is set to blink or not.

Return Value

Boolean	Returns TRUE if the color is set to blink. Returns FALSE if the function did not successfully set the color to blink, such as if no color is identified.

drGetColorBlink

Boolean

```
drGetColorBlink(  
    drDisplayId    displayId,  
    unsigned int   colorIndex,  
    Boolean        *pBlink  
);
```

Description

Returns the information on whether a specified color is set to blink.

Arguments

displayId	The identifier of the display.
colorIndex	The color index which uniquely identifies the color.
*pBlink	The state indicating whether the color is set to blink or not.

Return Value

Boolean

Returns TRUE if the display and color index are legal. If the function returns TRUE, the value of the argument pointer *pBlink has been updated. If its value is TRUE, the color is blinking.

Returns FALSE if the display or color index are invalid. If the value of the argument pointer is FALSE, the color does not blink. In this case, the value of *pBlink is undefined and should not be used.

Cadence Integrators Toolkit Database Reference

CDBA Display Resource File Functions

drGetColor

```
drColor *  
drGetColor(  
    drDisplayId  displayId,  
    unsigned int  colorIndex  
);
```

Description

The structure contents specified by the pointer returned is valid only until the next call to `drGetColor` or `drGetColorByName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>colorIndex</code>	The index uniquely identifying the color.

Return Value

<code>drColor *</code>	Returns the color from the color table index. The function returns NULL if the color index or display identifier are invalid or if the function is not successful.

drGetColorByName

```
drColor *  
drGetColorByName(  
    drDisplayId  displayId,  
    String       colorName  
);
```

Description

Returns the first color the color name refers to. The structure contents specified by the pointer returned is valid only until the next call to `drGetColor` or `drGetColorByName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>colorName</code>	The name of the color.

Return Value

<code>drColor *</code>	It returns NULL if the color display identifier is not defined or if not successful.

drGetColorIndexByName

```
int
drGetColorIndexByName (
    drDisplayId    displayId,
    String          colorName
);
```

Description

Returns the color index from the color name. Use this routine when you want to verify if a named color has been defined, without being interested in the color structure.

Arguments

displayId	The identifier of the display.
colorName	The name of the color.

Return Value

int	The function returns the color index if successful. Otherwise, it returns -1 if the color is not defined or if not successful.

drStartGenColor

```
drGenStateId  
drStartGenColor(  
    drDisplayId displayId  
);
```

Description

Starts the generator function for colors of a display.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>drGenStateId</code>	Returns a valid <code>drGenStateId</code> if the function is successfully completed. Returns NULL if not, or if the display is not successfully identified.

drGenColor

Boolean

```
drGenColor(  
    state, **packetPP  
);  
  
drGenStateId state;  
drColor **color;
```

Description

Generates the next color of a display.

Arguments

state	The generator state identifier returned by the <code>drStartGenColor</code> function.
**color	The next color for a display which is generated, returned by indirection.

Return Value

Boolean	The function returns FALSE if the generator state identifier is invalid, or if the generator is finished.

Functions to Generate DRM Colors

These generator functions generate all DRM (Display Resource Manager) colors associated with the specified display device.

```
drGenStateId pState;  
drDisplayId displayId;  
drColor *colorP;  
  
...  
pState = drStartGenColor(displayId);  
if ( !pState ) {  
    fprintf("DRM: Failed to generate colors.\n");  
    return FALSE;  
}  
while (drGenColor(pState, &colorP)) {  
    printf("Color Name %s\n", colorP->name);  
}  
drStopGen(pState);
```

Note: When you call a display resources generator to generate a display, color, stipple, linestyle, or packet, remember to call `drStopGen()` when finished, to properly terminate the generator.

Stipple Patterns

A stipple pattern fills shapes on a packet if the packet's fill style specifies to use a fill pattern. Stipple patterns are stored in the display's stipple table and they are addressed either by the stipple table index or the stipple name. The stipple table index is guaranteed to be unique and to remain the same during the life time of the stipple table. The stipple pattern parameters are specified in a structure of type `drStipple`. This section gives a structure and functions for stipple patterns.

drStipple

```
typedef struct drStipple drStipple;
    struct          drStipple {
    String          name;
    unsigned int    index;
    unsigned int    width;
    unsigned int    height;
    unsigned char   *pattern;
    };
```

Description

Creates a stipple pattern. Each element may have either the value 0 or the value 1. Given the index of a bit in the pattern, its row and column are calculated as:

row = index / width

column = index % width

When making a stipple pattern, initialize all fields.

Arguments

<code>drStipple</code>	The name of the structure.
<code>name</code>	The field <code>name</code> specifies the stipple name.
<code>index</code>	The field <code>index</code> specifies the index of the stipple pattern in the stipple table.
<code>width</code>	The field <code>width</code> specifies the width of the stipple pattern in pixels.
<code>height</code>	The field <code>height</code> specifies the height of the stipple pattern in pixels.
<code>pattern</code>	The field <code>pattern</code> is a pointer to a linear representation of the two dimensional stipple pattern, where each one byte element represents a bit in the stipple pattern.

drMakeStipple

Boolean

```
drMakeStipple(  
    drDisplayId  displayId,  
    drStipple    *pStipple  
);
```

Description

Makes a stipple pattern in the display's stipple table. If the stipple pattern does not exist, it is created. If the stipple pattern exists, its parameters are updated. The stipple index uniquely identifies the stipple pattern.

Arguments

displayId	The identifier of the display.
*pStipple	The pointer to the <code>drStipple</code> initialized data structure. it describes the stipple to create or update.

Return Value

Boolean	The function returns TRUE if the modification or creation is successful; otherwise, it returns FALSE.

drGetStipple

```
drStipple *  
drGetStipple(  
    drDisplayId  displayId,  
    unsigned int  stippleIndex  
);
```

Description

Returns the stipple pattern specified by the stipple index. The structure contents specified by the pointer returned are valid only until the next call to `drGetStipple` or `drGetStippleByName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>stippleIndex</code>	The stipple index.

Return Value

<code>drStipple *</code>	The function returns NULL if the stipple pattern is not defined, if the display identifier is not valid, or if the function is not successful.

drGetStippleByName

```
drStipple *  
drGetStippleByName(  
    drDisplayId  displayId,  
    String       stippleName  
);
```

Description

Returns the stipple pattern specified by the stipple name. The structure contents specified by the pointer returned are valid only until the next call to `drGetStipple` or `drGetStippleByName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>stippleName</code>	The stipple name.

Return Value

<code>drStipple *</code>	The function returns NULL if the stipple pattern is not defined, if the display identifier is invalid, or if the function is not successful.

drDeleteStippleByName

Boolean

```
drDeleteStippleByName(  
    drDisplayId    displayId,  
    String         stippleName  
);
```

Description

Deletes the stipple pattern specified by the stipple name.

Arguments

displayId	The identifier of the display.
stippleName	The stipple name.

Return Value

Boolean	The function returns TRUE if the stipple was deleted. The function returns FALSE if the display identifier or the stipple name is invalid, or if the stipple was not deleted.

drGetStippleIndexByName

```
int
drGetStippleIndexByName (
    drDisplayId  displayId,
    String       stippleName
);
```

Description

Returns the index of the stipple pattern specified by the stipple name. This routine is used when you want to verify if a named stipple pattern has been defined, without being interested in the pattern.

Arguments

displayId	The identifier of the display.
stippleName	The stipple name.

Return Value

int	The index of the stipple pattern specified by the stipple name. The function returns -1 if the stipple pattern is not defined or if not successful.

Cadence Integrators Toolkit Database Reference

CDBA Display Resource File Functions

drStartGenStipple

```
drGenStateId  
drStartGenStipple(  
    drDisplayId displayId  
);
```

Description

Prepares to generate all DRM stipples associated with the specified display device.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>drGenStateId</code>	The generator state identifier to be used in the <code>drGenStipple</code> function.

drGetNumberOfStipple

```
int  
drGetNumberOfStipple(  
    displayId  
);
```

Description

Retrieves the number of DRM (Display Resource Manager) stipples associated with the specified display device.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>int</code>	The number of DRM stipples returned.

drGenStipple

Boolean

```
drGenStipple(  
    drGenStateId  state,  
    drStipple     **stipple  
);
```

Description

Generates all DRM stipples associated with the specified display device.

Arguments

<code>state</code>	The generator state identifier returned by the <code>drStartGenStipple</code> function.
<code>**stipple</code>	The next stipple generated.

Return Value

Boolean	The function returns FALSE if the generator state identifier is invalid, if the stipples do not exist, or if the generator is finished. If the function returns TRUE, the stipple updated points to the <code>drStipple</code> data structure.

Line Styles

Line styles are stored in the display's line style table. They are addressed by either the line style table index or the line style name. The line style table index is guaranteed to be unique and to remain the same during the lifetime of the line style table. The line style parameters are specified in a structure of type `drLineStyle`.

This section gives a structure and functions for line styles.

drLineStyle

```
typedef struct drLineStyle drLineStyle;  
struct drLineStyle {  
    String      name;  
    unsigned int index;  
    unsigned int length;  
    unsigned int width;  
    unsigned char *pattern;  
};
```

Description

The line style data structure.

Arguments

name	The field <code>name</code> specifies the line style name.
index	The field <code>index</code> specifies the index of the line style in the line style table.
length	The field <code>length</code> specifies the length of the line style pattern in pixels.
width	The field <code>width</code> specifies the width of the line in pixels.
*pattern	The field <code>*pattern</code> is a pointer to an array of bytes where each one byte element represents a bit in the line style pattern.

drMakeLineStyle

Boolean

```
drMakeLineStyle(  
    drDisplayId    displayId,  
    drLineStyle    *pLineStyle  
);
```

Description

Makes a line style in the display's line style table. If the line style does not exist, it is created. If the line style exists, its parameters are updated. The line style index uniquely identifies the line style.

Arguments

displayId	The identifier of the display.
*pLineStyle	The pointer to the line style.

Return Value

Boolean	The function returns TRUE if the creation or modification is successful; otherwise, it returns FALSE.

drGetLineStyle

```
drLineStyle *  
drGetLineStyle(  
    drDisplayId    displayId,  
    unsigned int    lineStyleIndex  
);
```

Description

Returns the line style specified by the line style index. The structure contents specified by the pointer returned is valid only until the next call to `drGetLineStyle` or `drGetLineStyleByName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>lineStyleIndex</code>	The line style index.

Return Value

<code>drLineStyle *</code>	The line style specified by the line style index. The function returns NULL if the line style is not defined or if not successful.

drGetLineStyleByName

```
drLineStyle *  
drGetLineStyleByName (  
    drDisplayId    displayId,  
    String         lineStyleName  
);
```

Description

Returns the first line style specified by the line style name. The structure contents specified by the pointer returned is valid only until the next call to `drGetLineStyle` or `drGetLineStyleByName`.

Arguments

<code>displayId</code>	The identifier of the display.
<code>lineStyleName</code>	The line style name.

Return Value

<code>drLineStyle *</code>	The first line style specified by the line style name. The function returns NULL if the line style is not defined or if not successful.

drGetLineStyleIndexByName

```
int  
drGetLineStyleIndexByName(  
    drDisplayId    displayId,  
    String         lineStyleName  
);
```

Description

Returns the index of the line style specified by the line style name. This routine is used when you want to verify if a named line style has been defined, without being interested in the line style.

Arguments

displayId	The identifier of the display.
lineStyleName	The line style name.

Return Value

int	The index of the line style specified by the line style name. The function returns <code>-1</code> if the line style is not defined or if not successful.

drStartGenLineStyle

```
drGenStateId  
drStartGenLineStyle(  
    drDisplayId displayId  
);
```

Description

Starts the generator to generate the line styles in a display.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>drGenStateId</code>	The generator state identifier to be used in the <code>drGenLineStyle</code> function.

drGenLineStyle

Boolean

```
drGenLineStyle(  
    drGenStateId  pState,  
    drLineStyle** lineStyle  
);
```

Description

Generates the line styles in a display.

Arguments

pState	The generator state identifier returned by the drStartGenLineStyle function.
lineStyle **	The next line style generated.

Return Value

Boolean	If the function returns TRUE, the function is successful and the line style argument is assigned and valid. If the generator state identifier or the line style is invalid, or if the generator is finished, this function returns FALSE.

Packets

Packets are stored in the display's packet hash table. They are addressed by the packet name. This section gives a structure and functions for packets.

The packet parameters are specified in a structure of type `drPacket`.

```
drPacket
typedef struct drPacket drPacket;
struct drPacket {
String name;
unsigned int stipIdx;
unsigned int lineStyleIdx;
unsigned int fillIdx;
unsigned int outlineIdx;
};
```

Description

The value of `stipIdx` of a packet is `USHRT_MAX(65535)` if the user specifies `_NA_` or `_na_stipple` for a packet. The value of `lineStyleIdx` of a packet is `USHRT_MAX(65535)` if the user specifies `_NA_` or `_na_` for a packet. The value of `outlineIdx` is same as the value of `fillIdx` if the user specifies `_DEF_` or `_def_` for a packet.

Arguments

<code>name</code>	The field <code>name</code> specifies the packet name.
<code>stipIdx</code>	The field <code>stipIdx</code> specifies the index of the stipple in the stipple table.
<code>lineStyleIdx</code>	The field <code>lineStyleIdx</code> specifies the index of the line style in the line style table.
<code>fillIdx</code>	The field <code>fillIdx</code> specifies the index of the fill color in the color table.
<code>outlineIdx</code>	The field <code>outlineIdx</code> specifies the index of the outline color in the color table.

drMakePacket

Boolean

```
drMakePacket (
    drDisplayId  displayId
    drPacket     *pPacket
);
```

Description

Makes a packet in the display's packet hash table. If the packet does not exist, it is created. If the packet exists, its parameters are updated. The packet name uniquely identifies the packet.

Arguments

displayId	The identifier of the display.
*pPacket	The address of the packet.

Return Value

Boolean	The function returns TRUE if the modification is successful; otherwise, it returns FALSE.

drGetPacket

```
drPacket *  
drGetPacket(  
    drDisplayId    displayId,  
    String         PacketName  
);
```

Description

Returns the packet specified by the packet name.

Arguments

displayId	The identifier of the display.
PacketName	The specified packet name.

Return Value

drPacket *	The packet. The function returns NULL if the packet is not defined or if not successful.

drDeletePacket

```
void  
drDeletePacket(  
    drDisplayId    displayId,  
    String          PacketName  
);
```

Description

Deletes a packet in a display.

Arguments

displayId	The identifier of the display.
PacketName	The packet name.

Return Value

None.

drMakePacketAlias

Boolean

```
drMakePacketAlias(  
    drDisplayId    displayId,  
    String         aliasPacketName,  
    String         srcPacketName  
);
```

Description

Creates an alias for a packet name.

Arguments

displayId	The identifier of the display.
aliasPacketName	The alias for the packet name.
srcPacketName	The original packet name.

Return Value

Boolean	Returns TRUE if the alias is successfully created. Returns FALSE if the display or packet name is not successfully identified.

drGetNumberOfPackets

```
int  
drGetNumberOfPackets (  
    displayId  
);  
drDisplayId  displayId;
```

Description

Returns the number of packets in a display.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>int</code>	The number of packets in a display.

drGetPacketIndex

```
drPacketId  
drGetPacketIndex(  
    drDisplayId    displayId,  
    String         PacketName  
);
```

Description

Returns the identifier of a packet in a display.

Arguments

displayId	The identifier of the display.
PacketName	The packet name.

Return Value

drPacketId	The identifier of the packet.

drFindPacket

```
drPacket *  
drFindPacket(  
    drDisplayId  displayId,  
    String       PacketName  
);
```

Description

Locates a packet in a display.

Arguments

displayId	The identifier of the display.
PacketName	The packet name.

Return Value

drPacket *	The packet.

drMakeDefaultPacket

```
drPacket *  
drMakeDefaultPacket(  
    drDisplayId    displayId,  
    String         func  
);
```

Description

Creates a default packet in a display.

Arguments

displayId	The identifier of the display.
func	The name of the calling function.

Return Value

drPacket *	The default packet created.

drStartGenPacketIndex

```
drGenStateId  
drStartGenPacket(  
    drDisplayId displayId  
);
```

Description

Starts the generator to generate a packet index for a display.

Arguments

<code>displayId</code>	The identifier of the display.

Return Value

<code>drGenStateId</code>	The generator state identifier to be used in the <code>drGenPacket</code> function.

drGenPacket

Boolean

```
drGenPacket(  
    state, **packetPP  
);  
  
drGenStateId state;  
drPacket **packetPP;
```

Description

Generates all DRM (display resource management) packets associated with the specified display device.

Arguments

<code>state</code>	The generator state identifier returned by the <code>drStartGenPacket</code> function.
<code>**packetPP</code>	The display resource management packets to be generated.

Return Value

Boolean	If the DRM packets do not exist, the generator state identifier is invalid, or the generator is finished, this function returns FALSE.

CDBA Hierarchy Database Functions

The hierarchy database interface (HDB) is an interface to the Cadence Hierarchy Editor's configuration-driven elaboration process. Elaboration is the process of traversing the design hierarchy and using HDB to pick the right views for a specific purpose. Elaboration is also sometimes referred to as "design binding" or "traversal."

HDB reads the Hierarchy Editor's configuration file (`expand.cfg` file) and lets you write an elaboration routine.

Note: You cannot write to the `expand.cfg` file with this interface.

HDB consists of C functions that have the prefix `hdb`. These functions are listed in [HDB Functions](#).

An example traverser, provided with Cadence software, illustrates the usage of the HDB interface. The example traverser is in the following directory:

```
your_install_dir/tools/itkDB/samples/HDB
```

The key code is in the `traverse.c` file. This example binds a design and then prints various levels of information as it traverses the design.

A SKILL version of the example is in the following file:

```
your_install_dir/tools/hdb/examples/hdbTraverse.il
```

The SKILL example has more extensive comments than the C example. See the `hdbTraverse.il` file for more information.

HDB Functions

The HDB interface includes the following C functions. For more information about these functions, see the example referred to above as well as the `your_install_dir/tools/itkDB/include/itkDB.h` file.

```
hdbUserErrFunc hdbSetErrorHandler(hdbUserErrFunc func);  
void hdbDefaultErrorHandler(hdbErrorLevel level, char* message);  
char* hdbGetError();
```

Cadence Integrators Toolkit Database Reference

CDBA Hierarchy Database Functions

```
void hdbClearError();

hdbConfigId hdbOpen(const char* libName, const char* cellName,
                    const char* viewName, const char* mode);

hdbConfigId hdbOpenWithType(const char* libName, const char* cellName,
                             const char* viewName, const char* mode, const char* nameSpace);

hdbConfigBagId hdbAllocConfigBag();

Boolean hdbAddConfigToBag(hdbConfigBagId bag, hdbConfigId config);

Boolean hdbCloseConfigsInBag(hdbConfigBagId bagId);

Boolean hdbConfigHasOccRules(hdbConfigId config);

Boolean hdbSave(hdbConfigId config);

Boolean hdbSaveAs(hdbConfigId config, const char* libName, const char* cellName,
                  const char* viewName);

Boolean hdbIsOpenConfig(hdbConfigId config);

Boolean hdbClose(hdbConfigId config);

Boolean hdbConfigIsModified(hdbConfigId config);

Boolean hdbConfigIsEditable(hdbConfigId config);

char* hdbGetLibName(hdbConfigId config);

char* hdbGetCellName(hdbConfigId config);

char* hdbGetViewName(hdbConfigId config);

char* hdbGetTopLibName(hdbConfigId config);

char* hdbGetTopCellName(hdbConfigId config);

char* hdbGetTopViewName(hdbConfigId config);

Boolean hdbSetTopCellViewName(hdbConfigId config, const char* libName,
                              const char* cellName, const char* viewName);

char** hdbGetDefaultLibList(hdbConfigId config);

char* hdbGetDefaultLibListString(hdbConfigId config);

Boolean hdbSetDefaultLibListString(hdbConfigId config, const char* libListStr);

Boolean hdbSetDefaultLibList(hdbConfigId config, char** libList);

char** hdbGetDefaultViewList(hdbConfigId config);

char* hdbGetDefaultViewListString(hdbConfigId config);

Boolean hdbSetDefaultViewListString(hdbConfigId config, const char* viewListStr);

Boolean hdbSetDefaultViewList(hdbConfigId config, char** viewList);

char** hdbGetDefaultStopList(hdbConfigId config);

char* hdbGetDefaultStopListString(hdbConfigId config);

Boolean hdbSetDefaultStopListString(hdbConfigId config, const char* stopListStr);

Boolean hdbSetDefaultStopList(hdbConfigId config, char** stopList);

Boolean hdbDeleteNote(hdbConfigId config);

Boolean hdbReplaceNoteLine(hdbConfigId config, const char* text);

Boolean hdbAddNoteLine(hdbConfigId config, const char* text);

char* hdbGetNoteString(hdbConfigId config);

Boolean hdbReplaceConst(hdbConfigId config, const char* name, const char* value);

Boolean hdbReplaceConstArr(hdbConfigId config, const char* name, char** valueArr);
```

Cadence Integrators Toolkit Database Reference

CDBA Hierarchy Database Functions

```
Boolean hdbDeleteConst(hdbConfigId config, const char* name);
int hdbGetRevision(hdbConfigId config);
Boolean hdbIsConfigModified(hdbConfigId config);
char** hdbGetObjLibList(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
char* hdbGetObjLibListStr(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
Boolean hdbSetObjLibListStr(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName,
    const char* libListStr);
Boolean hdbSetObjLibList(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName,
    char** libList);
Boolean hdbDeleteObjLibList(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName );
char** hdbGetObjViewList(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
char* hdbGetObjViewListStr(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
Boolean hdbSetObjViewListStr(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName,
    const char* viewListStr);
Boolean hdbSetObjViewList(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName,
    char** viewList);
Boolean hdbDeleteObjViewList(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
hdbSpecId hdbGetObjBinding(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
Boolean hdbSetObjBinding(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName,
    const char* libBinding, const char* cellBinding,
    const char* viewBinding);
Boolean hdbDeleteObjBinding(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
Boolean hdbGetObjStop(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
Boolean hdbSetObjStop(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName,
    Boolean doStop);
Boolean hdbDeleteObjStop(hdbConfigId config, const char* libName,
    const char* cellName, const char* viewName, const char* instName);
hdbPathVectorId hdbCreatePathVector(hdbConfigId config);
hdbPathVectorId hdbCopyPathVector(hdbPathVectorId pathVector);
void hdbDestroyPathVector(hdbPathVectorId pathVector);
Boolean hdbPop(hdbPathVectorId pathVector);
Boolean hdbPopCell(hdbPathVectorId pathVector);
Boolean hdbPopToTop(hdbPathVectorId pathVector);
```

Cadence Integrators Toolkit Database Reference

CDBA Hierarchy Database Functions

```
Boolean hdbPushCell(hdbPathVectorId pathVector, const char* instName,
                    const char* libName, const char* cellName, const char* viewName);

Boolean hdbPushCell2(hdbPathVectorId pathVector, const char* instName,
                    const char* libName, const char* cellName, const char* viewName,
                    hdbBindWhyId bindWhy);

char* hdbGetPathStr(hdbPathVectorId pathVector);

char* hdbGetPathStackStr(hdbPathVectorStackId pathVecStack);

hdbSpecId hdbBind(hdbPathVectorId pathVector, const char* libName,
                  const char* cellName, const char* viewName, const char* instName,
                  Boolean libFixed, Boolean cellFixed, Boolean viewFixed,
                  hdbSpecId rtnSpec);

hdbSpecId hdbBind2(hdbPathVectorId pathVector, const char* libName,
                  const char* cellName, const char* viewName, const char* instName,
                  Boolean libFixed, Boolean cellFixed, Boolean viewFixed,
                  hdbSpecId rtnSpec, hdbBindSignatureId* rtnBindSig);

hdbSpecId hdbBind3(hdbPathVectorId pathVector, const char* libName,
                  const char* cellName, const char* viewName, const char* instName,
                  Boolean libFixed, Boolean cellFixed, Boolean viewFixed,
                  hdbSpecId rtnSpec, hdbBindSignatureId* rtnBindSig,
                  hdbBindWhyId* rtnBindWhy);

Boolean hdbIsAtStopPoint(hdbPathVectorId pathVec);

char* hdbExplainStop(const char* indent, hdbPathVectorId pathVec);

Boolean hdbEquivalentSigs(hdbBindSignatureId sig1, hdbBindSignatureId sig2);

int hdbHashSig(hdbBindSignatureId sig);

char* hdbExplainBinding(const char* indent, const char* libName,
                        const char* cellName, const char* viewName, const char* instName,
                        Boolean libFixed, Boolean cellFixed, Boolean viewFixed,
                        hdbSpecId bindingSpec, hdbBindSignatureId bindSig,
                        hdbBindWhyId bindWhy);

char* hdbExplainSignature(const char* indent, hdbBindSignatureId bindSig);

hdbPathVectorStackId hdbCreatePathVectorStack();

void hdbPushPathVect(hdbPathVectorStackId pathVectStack,
                    hdbPathVectorId pathVector, void* userData);

Boolean hdbGetPathVectorStackTop(hdbPathVectorStackId pathVectStack,
                                hdbPathVectorId* rtnPathVectorIdPt, void** rtnUserDataPt);

void hdbPopPathVect(hdbPathVectorStackId pathVectStack);

Boolean hdbIsConfig(hdbSpecId spec);

Boolean hdbIsOpenBinding(hdbSpecId spec);
```

Related Information

For more information about the Hierarchy Editor's elaboration process, see the [*Cadence Hierarchy Editor User Guide*](#).

Cadence Integrators Toolkit Database Reference

CDBA Hierarchy Database Functions

Cadence Integrators Toolkit Database Reference

CDBA Hierarchy Database Functions

CST Access C Functions

This chapter describes the CST Access C API's.

For more information, see the following topics:

- [Constraint Functions](#) on page 1650
- [Constraint Group Functions](#) on page 1679
- [Generator Functions](#) on page 1692
- [20nm CST APIs](#) on page 1706

Constraint Functions

This section describes the following constraint functions.

- [cstCreateLayerConstraint](#)
- [cstCreateLayerPairConstraint](#)
- [cstCreateLayerArrayConstraint](#)
- [cstCreateSimpleConstraint](#)
- [cstAddToConstraintGroup](#)
- [cstReplaceIdsInConstraintGroup](#)
- [cstDeleteConstraint](#)
- [cstGetType](#)
- [cstGetConstraintDatabase](#)
- [cstGetConstraintDefName](#)
- [cstGetConstraintName](#)
- [cstGetConstraintDescription](#)
- [cstGetConstraintID](#)
- [cstGetLayerConstraintLayer](#)
- [cstGetLayerPairConstraintLayers](#)
- [cstGetLayerArrayConstraintLayers](#)
- [cstGetConstraintValue](#)
- [cstSetConstraintValue](#)
- [cstGetConstraintParams](#)
- [cstSetConstraintParams](#)
- [cstIsHardConstraint](#)
- [cstSetHardConstraint](#)
- [cstSetConstraintID](#)
- [cstSetConstraintDescription](#)

cstCreateLayerConstraint

```
cstLayerConstraintId
cstCreateLayerConstraint (
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    cstConstraintDescriptor descriptor,
    cstLayerNum             layerNum,
    cstPurpose              purpose,
    dbPropType              valueType,
    dbPropValue             value,
    dbPropType              paramListType,
    dbPropValue             listParams,
    cppConstString          constraintName
);
```

Description

Creates a layer constraint in the constraint group (cstConstraintGroupId) with the specified constraint definition name (constraintDefName), layers, constraint value, constraint parameters, hard attribute (descriptor), the position of the constraint with the members of the constraint group (descriptor) and the name of the constraint.

Arguments

<i>groupId</i>	The ID of the constraintGroup, which is to own this constraint.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>descriptor</i>	The descriptor can be a cstcHardConstraintDescriptor, a cstcAppendConstraintDescriptor, or both (cstcHardConstraintDescriptor cstcAppendConstraintDescriptor). If cstcHardConstraintDescriptor is not specified then the constraint is a "soft" constraint. If cstcAppendConstraintDescriptor is not specified then the constraint will be put to the front of the member list.
<i>layerNum</i>	The layer number of the constraint. If the layer number is unknown, an error results
<i>purpose</i>	The purpose number of the layer constraint.

Cadence Integrators Toolkit Database Reference

CST Access C Functions

<code>valueType</code>	The type of value for the layer constraint.
<code>value</code>	The value for the layer constraint.
<code>paramListType</code>	A database constant variable, which is <code>dbclLListType</code> .
<code>listParams</code>	A SKILL list of constraint parameters. The default value is <code>ilcNil</code> . The input <code>constraintParams</code> is in a format of <code>list(list('paramKeyWord{int float string list}))...)</code> For example, <code>list(list('ConnectivityType "sameNet") list('PGNet t))</code>
<code>constraintName</code>	The name of the layer constraint.

Return Value

<code>cstLayerConstraintId</code>	Returns the <code>cstLayerConstraintId</code> of the created layer constraint.

cstCreateLayerPairConstraint

```
cstLayerPairConstraintId
cstCreateLayerPairConstraint (
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    cstConstraintDescriptor descriptor,
    cstLayerNum             layerNum1,
    cstPurpose              purpose1,
    cstLayerNum             layerNum2,
    cstPurpose              purpose2,
    dbPropType              valueType,
    dbPropValue             value,
    dbPropType              paramListType,
    dbPropValue             listParams,
    cppConstString          constraintName
);
```

Description

Creates a layer-purpose pair constraint in the constraint group (cstConstraintGroupId) with the specified constraint definition name (constraintDefName), layers, constraint value, constraint parameters, hard attribute (descriptor), the position of the constraint with the members of the constraint group (descriptor) and the name of the constraint.

Arguments

<i>groupId</i>	The ID of the constraintGroup, which is to own this constraint.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>descriptor</i>	The descriptor can be a cstcHardConstraintDescriptor, a cstcAppendConstraintDescriptor, or both (cstcHardConstraintDescriptor cstcAppendConstraintDescriptor). If cstcHardConstraintDescriptor is not specified then the constraint is a "soft" constraint. If cstcAppendConstraintDescriptor is not specified then the constraint will be put to the front of the member list.
<i>layerNum1</i>	The layer number of the first layer-purpose pair constraint. If the layer number is unknown, an error results

cstCreateLayerArrayConstraint

```
cstLayerArrayConstraintId
cstCreateLayerArrayConstraint (
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    cstConstraintDescriptor descriptor,
    unsigned int            numLayers,
    cstLayerNum              layerNums,
    cstPurpose               purposes,
    dbPropType               valueType,
    dbPropValue              value,
    dbPropType               paramListType,
    dbPropValue              listParams,
    cppConstString          constraintName
);
```

Description

Creates a layer array constraint in the constraint group (cstConstraintGroupId) with the specified constraint definition name (constraintDefName), layers, constraint value, constraint parameters, hard attribute(descriptor), the position of the constraint with the members of the constraint group (descriptor) and the name of the constraint.

Arguments

<i>groupId</i>	The ID of the constraintGroup, which is to own this constraint.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>descriptor</i>	The descriptor can be a cstcHardConstraintDescriptor, a cstcAppendConstraintDescriptor, or both (cstcHardConstraintDescriptor cstcAppendConstraintDescriptor). If cstcHardConstraintDescriptor is not specified then the constraint is a "soft" constraint. If cstcAppendConstraintDescriptor is not specified then the constraint will be put to the front of the member list.
<i>numLayers</i>	The number of layers in the array of layerNums

Cadence Integrators Toolkit Database Reference

CST Access C Functions

<i>layerNums</i>	The layer number of the array constraint. An array contains layer numbers. If the layer number is unknown, an error results.
<i>purposes</i>	The purpose number of the layer array constraint. An array contains purpose numbers.
<i>valueType</i>	The type of value for the layer array constraint.
<i>value</i>	The value for the layer array constraint.
<i>paramListType</i>	A database constant variable, which is dbclLListType.
<i>listParams</i>	A SKILL list of constraint parameters. The default value is ilcNil. The input constraintParams is in a format of list(list('paramKeyWord{int float string list}))...) For example, list(list('ConnectivityType "sameNet") list('PGNet t))
<i>constraintName</i>	The name of the layer array constraint.

Return Value

<i>cstLayerArrayConstraintId</i>	Returns the constraint ID created with the layer or layer-purpose pair array.

cstCreateSimpleConstraint

```
cstSimpleConstraintId
cstCreateSimpleConstraint (
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    cstConstraintDescriptor descriptor,
    dbPropType              valueType,
    dbPropValue             value,
    dbPropType              paramListType,
    dbPropValue             listParams,
    cppConstString          constraintName
);
```

Description

Creates a simple constraint in the constraint group (cstConstraintGroupId) with the specified constraint definition name (constraintDefName), layers, constraint value, constraint parameters, hard attribute (descriptor), the position of the constraint with the members of the constraint group (descriptor) and the name of the constraint.

Arguments

<i>groupId</i>	The ID of the constraintGroup, which is to own this constraint.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>descriptor</i>	The descriptor can be a cstcHardConstraintDescriptor, a cstcAppendConstraintDescriptor, or both (cstcHardConstraintDescriptor cstcAppendConstraintDescriptor). If cstcHardConstraintDescriptor is not specified then the constraint is a "soft" constraint. If cstcAppendConstraintDescriptor is not specified then the constraint will be put to the front of the member list.
<i>valueType</i>	The type of value for the simple constraint.
<i>value</i>	The value for the simple constraint.
<i>paramListType</i>	A database constant variable, which is dbcILLListType.

Cadence Integrators Toolkit Database Reference

CST Access C Functions

listParams A SKILL list of constraint parameters. The default value is `ilcNil`. The input `constraintParams` is in a format of `list(list('paramKeyWord{int | float | string | list}))...`
For example, `list(list('ConnectivityType
"sameNet")
list('PGNet t))`

constraintName The name of the simple constraint.

Return Value

cstSimpleConstraintId Returns the *cstSimpleConstraintId* of the created simple constraint.

cstAddToConstraintGroup

Boolean

```
cstAddToConstraintGroup(  
    cstConstraintGroupId  groupId,  
    cstConstraintGroupId  memberGroupId,  
    Boolean               append  
);
```

Description

Adds a constraint group ID (`cstMemberConstraintGroupID`) to be a member constraint group of the containing constraint. You can position the constraint group at the beginning of the members or at the end of the members with `append`. The default value for `append` is `nil`, which means that add the constraint group to the beginning of the members. The member `constraintGroup` cannot be owned by other objects.

Arguments

<i>groupId</i>	The ID of the <code>constraintGroup</code> , which is to own this constraint.
<i>memberGroupID</i>	The ID of the member constraint group.
<i>append</i>	A boolean having a value as either 'nil' or 't'. Having the value as 'nil' puts this constraint to the first member in the constraint group. The present first constraint group member is moved to the second and subsequent members are also moved down in order. However, if the value is 't', then the constraint is put to the end of the member in the constraint group.

Return Value

<i>Boolean</i>	Returns <code>TRUE</code> when <code>cstAddtoConstraintGroup</code> successfully adds a constraint to a constraint group. Returns <code>FALSE</code> if member constraint cannot be added to a constraint group.

cstReplaceIdsInConstraintGroup

Boolean

```
cstReplaceIdsInConstraintGroup(  
    cstConstraintGroupId  groupId,  
    unsigned int          numObjects,  
    cstId                 *Ids  
);
```

Description

You can set numObjects = 0 and Ids = NULL to remove the memberships of the constraints and member constraintGroups from the specified constraintGroup. If numObjects is the number of Ids(IDs), the Ids will become members of the constraintGroup.

Arguments

<i>groupId</i>	The ID of the constraintGroup, which is to own this constraint.
<i>numObjects</i>	The number of Ids (constraint IDs and/or constraintGroup IDs).
<i>*Ids</i>	The array contains constraint IDs and/or constraintGroup IDs.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when <code>Ids</code> are successfully replaced into the constraint group. Returns <code>FALSE</code> if the <code>Ids</code> cannot be replaced into the constraint group .

cstDeleteConstraint

Boolean

```
cstDeleteConstraint(  
    cstConstraintId constraintId  
);
```

Description

Deletes the constraint ID (*constraintId*) from any of the constraint groups it is a part of. It's value and constraint parameters are also deleted.

Arguments

<i>constraintId</i>	The ID of the constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when <code>cstDeleteConstraint</code> successfully deletes the constraint. Returns <code>FALSE</code> when the pass-in constraint cannot be deleted.

cstGetType

```
cstType  
cstGetType(  
    cstId Id  
);
```

Description

This function returns the type of the constraint.

Arguments

<i>Id</i>	The ID of the constraint.

Return Value

<i>cstType</i>	The valid values for the <i>cstType</i> that can be returned are: <pre>cstcUnknownType = 0, cstcLayerConstraintType = 2, cstcLayerPairConstraintType = 3, cstcLayerArrayConstraintType = 4, cstcSimpleConstraintType = 5, cstcConstraintGroupType = 6</pre>

cstGetConstraintDatabase

```
dbId  
cstGetConstraintDatabase(  
    cstConstraintId constraintId  
);
```

Description

This function returns the identification of the constraint database. The database can either be dbCellViewId or techFile.

Arguments

<i>constraintId</i>	The ID of the constraint.

Return Value

<i>dbID</i>	Returns the ID of the constraint database.

cstGetConstraintDefName

String

```
cstGetConstraintDefName(  
    cstConstraintId constraintId  
);
```

Description

This function returns the name of the mapped DFII constraint that is used to get a corresponding oaConstraintDef. For example, the defName can be "minSpacing", "validLayer", "validVias", etc.

Arguments

<i>constraintId</i>	The ID of the constraint.

Return Value

<i>string</i>	The name of the mapped dfll constraint.

cstGetConstraintName

String

```
cstGetConstraintName(  
    cstConstraintId constraintId  
);
```

Description

This function returns the name of the constraint. If a name was not specified while creation, it is automatically generated.

Arguments

<i>constraintId</i>	The ID of the constraint.

Return Value

<i>string</i>	The name of the constraint.

cstGetConstraintDescription

String

```
cstGetConstraintDescription(  
    cstConstraintId constraintId  
);
```

Description

This function returns the description of the constraint.

Arguments

<i>constraintId</i>	The ID of the constraint.

Return Value

<i>string</i>	The description of the constraint.

cstGetConstraintID

String

```
cstGetConstraintId(  
    cstConstraintId constraintId  
);
```

Description

This function returns the identification of the constraint.

Arguments

<i>constraintId</i>	The string ID of the constraint.

Return Value

<i>string</i>	The identification of the constraint.

cstGetLayerConstraintLayer

Boolean

```
cstGetLayerConstraintLayer(  
    cstLayerConstraintId  constraintId,  
    cstLayerNum           *layerNum,  
    cstPurpose            *purpose  
);
```

Description

This function gets the layer number of the layer associated with the specified constraint.

Arguments

<i>constraintId</i>	The ID of the layer constraint.
<i>layerNum</i>	The layer number of the constraint.
<i>purpose</i>	The purpose number of the constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when the layer number and purpose number are valid. Returns <code>FALSE</code> when the layer number and purpose number are invalid.

cstGetLayerPairConstraintLayers

Boolean

```
cstGetLayerPairConstraintLayers (
    cstLayerPairConstraintId  constraintId,
    cstLayerNum               *layer1Num,
    cstPurpose                *purpose1,
    cstLayerNum               *layer2Num,
    cstPurpose                *purpose2
);
```

Description

This function gets the layer number of the layer associated with the specified layer-purpose pair constraint.

Arguments

<i>constraintId</i>	The ID of the layer constraint.
<i>*layer1Num</i>	The layer number of the first layer-purpose pair constraint.
<i>*purpose1</i>	The purpose number of the first layer-purpose pair constraint.
<i>*layer2Num</i>	The layer number of the second layer-purpose pair constraint.
<i>*purpose2</i>	The purpose number of the second layer-purpose pair constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when the layer number and purpose number are valid. Returns <code>FALSE</code> when the layer number and purpose number are invalid.

cstGetLayerArrayConstraintLayers

Boolean

```
cstGetLayerArrayConstraintLayers (  
    cstLayerArrayConstraintId  constraintId,  
    unsigned int               *numOfLayers,  
    cstLayerNum                *layerNums,  
    cstPurpose                  *purposeNums  
);
```

Description

This function gets the layer number of the layer associated with the specified array constraint.

Arguments

<i>constraintId</i>	The ID of the layer constraint.
<i>*numOfLayers</i>	The number of array constraint layers.
<i>*layerNums</i>	The layer number of the array constraint.
<i>*purposeNums</i>	The purpose number of the array constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when the layer number and purpose number are valid. Returns <code>FALSE</code> when the layer number and purpose number are invalid.

cstGetConstraintValue

Boolean

```
cstGetConstraintValue(  
    cstConstraintId  constraintId,  
    dbPropType       valueType,  
    dbPropValue      value  
);
```

Description

This function returns the value of the specified constraint.

Arguments

<i>constraintId</i>	The ID of the constraint.
<i>valueType</i>	The type of the constraint value.
<i>value</i>	The value of the constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when value and valueType are successfully returned. Returns <code>FALSE</code> when the function is unsuccessful.

cstSetConstraintValue

Boolean

```
cstSetConstraintValue(  
    cstConstraintId  constraintId,  
    dbPropType       valueType,  
    dbPropValue      value  
);
```

Description

This function sets the value of the specified constraint. The old value is destroyed.

Arguments

<i>constraintId</i>	The ID of the constraint.
<i>valueType</i>	The type of the constraint value.
<i>value</i>	The value to be set for the constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when value and valueType are successfully set. Returns <code>FALSE</code> when the function is unsuccessful.

cstGetConstraintParams

Boolean

```
cstGetConstraintParams(  
    cstConstraintId  constraintId,  
    dbPropType       *paramListType,  
    dbPropValue      *listParams  
);
```

Description

This function returns the parameters associated with the specified constraint.

Arguments

<i>constraintId</i>	The ID of the constraint.
<i>*paramListType</i>	A database constant variable, which is dbcLLListType.
<i>*listParams</i>	A SKILL list of constraint parameters. The input constraintParams is in a format of <pre>list(list('paramKeyword{int float string list}))...</pre> For example, <code>list(list('ConnectivityType "sameNet") list('PGNet t))</code>

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when list of constraint parameters and list of types of constraint parameters are successfully returned. Returns <code>FALSE</code> when the function is unsuccessful.

cstSetConstraintParams

Boolean

```
cstSetConstraintParams(  
    cstConstraintId  constraintId,  
    dbPropType       paramListType,  
    dbPropValue      listParams  
);
```

Description

This function sets the parameters for the specified constraint. The old parameters are destroyed.

Arguments

<i>constraintId</i>	The ID of the constraint.
<i>paramListType</i>	A database constant variable, which is <code>dbclLListType</code> .
<i>listParams</i>	A SKILL list of constraint parameters. The input <code>constraintParams</code> is in a format of <code>list(list('paramKeyword{int float string list}))...</code> For example, <code>list(list('ConnectivityType "sameNet") list('PGNet t))</code>

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when list of constraint parameters and list of types of constraint parameters are successfully set. Returns <code>FALSE</code> when the function is unsuccessful.

cstIsHardConstraint

Boolean

```
cstIsHardConstraint(  
    cstconstraintId constraint  
);
```

Description

Returns a boolean value indicating that the constraint must be met. If `cstcHardConstraintDescriptor` is not specified then the constraint is a "soft" constraint. However, if a `cstcAppendConstraintDescriptor` is not specified then the constraint will be put to the front of the member list.

Arguments

<i>constraint</i>	The ID for <code>layerConstraint</code> , <code>layerPairCostraint</code> , <code>layerArrayConstraint</code> , and simple constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when it is a hard constraint. Returns <code>FALSE</code> when it is not a hard constraint.

cstSetHardConstraint

Boolean

```
cstSetHardConstraint(  
    cstConstraintId  constraint,  
    Boolean          hard  
);
```

Description

This function sets a boolean attribute on the constraint indicating whether or not it is a hard constraint.

Arguments

<i>constraint</i>	The ID for layerConstraint, layerPairCostraint, layerArrayConstraint, and simple constraint.
<i>hard</i>	A boolean indicating whether the constraint must be met. If the value is 'nil', it means that it is a soft constraint. By default, the value is 't'.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when hard constraint is successfully set. Returns <code>FALSE</code> when the function is unsuccessful.

cstSetConstraintID

Boolean

```
cstSetConstraintId(  
    cstConstraintId  constraint,  
    cppConstString  ID  
);
```

Description

This function sets the identification of the constraint.

Arguments

<i>constraint</i>	The ID for layerConstraint, layerPairCostraint, layerArrayConstraint, and simple costraint.
<i>ID</i>	The string id to be set on this constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when the ID for the constraint is successfully set. Returns <code>FALSE</code> when the function is unsuccessful.

cstSetConstraintDescription

Boolean

```
cstSetConstraintDescription(  
    cstConstraintId  constraint,  
    cppConstString  description  
);
```

Description

This function sets the description of the constraint.

Arguments

<i>constraint</i>	The ID for layerConstraint, layerPairCostraint, layerArrayConstraint, and simple costraint.
<i>description</i>	The string description to be set on the constraint.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> when the description for the constraint is successfully set. Returns <code>FALSE</code> when the function is unsuccessful.

Constraint Group Functions

This section describes the following constraint group functions.

- [cstCreateConstraintGroupOn](#)
- [cstFindConstraintGroupOn](#)
- [cstCreateConstraintGroupInTech](#)
- [cstCreateConstraintGroupInCellview](#)
- [cstFindConstraintGroupInTech](#)
- [cstFindConstraintGroupInCellview](#)
- [cstGetConstraintGroupOwner](#)
- [cstGetConstraintGroupDatabase](#)
- [cstDeleteConstraintGroup](#)
- [cstAccessGetConstraintGroupName](#)
- [cstGetConstraintGroupDefName](#)
- [cstGetConstraintGroupOperator](#)

cstCreateConstraintGroupOn

```
cstConstraintGroupId  
cstCreateConstraintGroupOn(  
    dbId                objectId,  
    cppConstString      groupDefName,  
    cstConstraintGroupOp op  
);
```

Description

Creates a constraint group ID on an object(e.g. net, term, etc) with arguments of the object ID, constraint group definition name, and operator symbol.

Cadence Integrators Toolkit Database Reference

CST Access C Functions

Arguments

<i>objectID</i>	The cellview object IDs applied with corresponding constraint group definition name.								
<i>groupDefName</i>	The DFII constraint group definition name. For more information, refer to the CST SKILL API documentation.								
<i>op</i>	The valid values of constraint group operator are: <table><tr><td><code>cstcUnknownConstraintGroupOp</code></td><td><code>= 0,</code></td></tr><tr><td><code>cstcPrecedenceConstraintGroupOp</code></td><td><code>= 1,</code></td></tr><tr><td><code>cstcAndConstraintGroupOp</code></td><td><code>= 2,</code></td></tr><tr><td><code>cstcOrConstraintGroupOp</code></td><td><code>= 3</code></td></tr></table>	<code>cstcUnknownConstraintGroupOp</code>	<code>= 0,</code>	<code>cstcPrecedenceConstraintGroupOp</code>	<code>= 1,</code>	<code>cstcAndConstraintGroupOp</code>	<code>= 2,</code>	<code>cstcOrConstraintGroupOp</code>	<code>= 3</code>
<code>cstcUnknownConstraintGroupOp</code>	<code>= 0,</code>								
<code>cstcPrecedenceConstraintGroupOp</code>	<code>= 1,</code>								
<code>cstcAndConstraintGroupOp</code>	<code>= 2,</code>								
<code>cstcOrConstraintGroupOp</code>	<code>= 3</code>								

Return Value

<i>cstConstraintGroupID</i>	The ID of the constraint group, which is created in the technology database or a cellview.

cstFindConstraintGroupOn

```
cstConstraintGroupId  
cstFindConstraintGroupOn(  
    dbId          objectId,  
    cppConstString groupDefName  
);
```

Description

Returns a constraint group ID (`constraintGroupId`) from an object (`object`) with constraint group definition name. The constraint value is required while `constraintParams` are optional for this function.

Arguments

<i>objectId</i>	The cellview object IDs applied with corresponding constraint group definition name.
<i>groupDefName</i>	The DFII constraint group definition name. For more information, refer to the CST SKILL API documentation.

Return Value

<i>cstConstraintGroupId</i>	The ID of the constraint group, which is created in the technology database or a cellview.

cstCreateConstraintGroupInTech

```
cstConstraintGroupId  
cstCreateConstraintGroupInTech(  
    cstTechFileId      techfileId,  
    cppConstString     groupName,  
    cppConstString     groupDefName,  
    cstConstraintGroupOp op  
);
```

Description

Creates a constraint group in the technology database (i.e techDb) with arguments of the database ID, constraint group name, constraint group definition name and operator symbol.

Arguments

<i>techfileId</i>	The database identifier of the technology file.
<i>groupName</i>	The name of the constraint group that needs to be specified by the user.
<i>groupDefName</i>	The DFII constraint group definition name can be "userDefined", "foundry", "implicit", "default", "cutClass".
<i>op</i>	The valid values of constraint group operator are: cstcUnknownConstraintGroupOp = 0, cstcPrecedenceConstraintGroupOp = 1, cstcAndConstraintGroupOp = 2, cstcOrConstraintGroupOp = 3

Return Value

<i>cstConstraintGroupId</i>	The ID of the constraint group, which is created in the technology database or a cellview.

cstCreateConstraintGroupInCellview

```
cstConstraintGroupId  
cstCreateConstraintGroupInCellview(  
    dbCellViewId      cellViewId,  
    cppConstString     groupName,  
    cppConstString     groupDefName,  
    cstConstraintGroupOp op  
);
```

Description

This function creates a constraint group attached to a specified cellview of `dbCellViewId`.

Arguments

<i>CellViewId</i>	The database identifier of the cellview.
<i>groupName</i>	The name of the constraint group that needs to be specified by the user.
<i>groupDefName</i>	The DFII constraint group definition name can be "userDefined", "implicit", and "default".
<i>op</i>	The valid values of constraint group operator are: cstcUnknownConstraintGroupOp = 0, cstcPrecedenceConstraintGroupOp = 1, cstcAndConstraintGroupOp = 2, cstcOrConstraintGroupOp = 3

Return Value

<i>cstConstraintGroupId</i>	The ID of the constraint group, which is created in the technology database or a cellview.

cstFindConstraintGroupInTech

```
cstConstraintGroupId  
cstFindConstraintGroupInTech(  
    cstTechFileId    techfileId,  
    cppConstString   groupName,  
    Boolean          localOnly  
);
```

Description

This function returns a constraint group ID (constraintGroupId) from a database ID (database) with constraint group name and a boolean flag (localOnly).

Arguments

<i>techfileId</i>	The database identifier of the technology file.
<i>groupName</i>	The name of the constraint group that needs to be specified by the user.
<i>localOnly</i>	Looks for a constraint group in an ITDB graph if the database ID is a technology database.

Return Value

<i>cstConstraintGroupId</i>	The ID of the constraint group, which is created in the technology database or a cellview.

cstFindConstraintGroupInCellview

```
cstConstraintGroupId  
cstFindConstraintGroupInCellview(  
    dbCellViewId    cellViewId,  
    cppConstString  groupName  
);
```

Description

This function returns a constraint group attached to a specified cellview of `dbCellViewId`.

Arguments

<i>CellViewId</i>	The database identifier of the cellview.
<i>groupName</i>	The name of the constraint group that needs to be specified by the user.

Return Value

<i>cstConstraintGroupId</i>	The ID of the constraint group, which is created in the technology database or a cellview.

cstGetConstraintGroupOwner

dbId

```
cstGetConstraintGroupOwner(  
    cstConstraintGroupId group  
);
```

Description

For a constraint group associated with a particular owning object, this function returns a pointer to that owning object.

Arguments

group

The ID of the constraint group, which is created in the technology database or a cellview.

Return Value

dbId

The database *dbId* for any database object.

cstGetConstraintGroupDatabase

```
dbId  
cstGetConstraintGroupDatabase(  
    cstConstraintGroupId group  
);
```

Description

This function returns database identifier (either techFileId or cellViewId) of the specified constraint group.

Arguments

<i>group</i>	The ID of the constraint group, which is created in the technology database or a cellview.

Return Value

<i>dbId</i>	The database <code>dbId</code> for any database object.

cstDeleteConstraintGroup

Boolean

```
cstDeleteConstraintGroup(  
    cstConstraintGroupId groupId  
);
```

Description

Deletes the constraint group (constraintGroupId). The contained constraints in the constraint group are not automatically deleted when the constraintGroup is deleted. You can delete constraints which are only owned by the specified constraintGroup.

Arguments

<i>groupId</i>	The ID of the constraint group that is to be deleted.

Return Value

<i>Boolean</i>	Returns <code>TRUE</code> when the constraint group is successfully deleted. Returns <code>FALSE</code> when unable to find a constraint group that is to be deleted.

cstAccessGetConstraintGroupName

String

```
cstAccessGetConstraintGroupName(  
    cstConstraintGroupId groupId  
);
```

Description

This function returns the name of the constraint group.

Arguments

<i>groupId</i>	The ID of the constraint group.

Return Value

<i>String</i>	Returns the name of the constraint group.

cstGetConstraintGroupDefName

String

```
cstGetConstraintGroupDefName(  
    cstConstraintGroupId groupId  
);
```

Description

This function gets the definition name of the DFII constraint group that is used to get a corresponding oaConstraintGroupDef. The DFII constraint group definition names can be "userDefined", "foundry", "implicit", "default", "taper", "inputTaper", "outputTaper", "shielding", transReflexive", "reflexive", "interChild", "cutClass".

Arguments

<i>groupId</i>	The ID of the constraint group.

Return Value

<i>String</i>	Returns the definition name of the DFII constraint group.

cstGetConstraintGroupOperator

```
cstConstraintGroupOp  
cstGetConstraintGroupOperator(  
    cstConstraintGroupId groupId  
);
```

Description

This function returns the constraint group operator associated with the constraint group.

Arguments

<i>groupId</i>	The ID of the constraint group.

Return Value

<i>cstConstraintGroupOp</i>	Returns the following valid values: cstcUnknownConstraintGroupOp = 0, cstcPrecedenceConstraintGroupOp = 1, cstcAndConstraintGroupOp = 2, cstcOrConstraintGroupOp = 3

Generator Functions

This section describes the following functions for generating constraints.

- [cstStartGenConstraintGroupTold](#)
- [cstGenConstraintGroupTold](#)
- [cstStartGenTechConstraintGroup](#)
- [cstGenTechConstraintGroup](#)
- [cstStartGenCellViewConstraintGroup](#)
- [cstGenCellViewConstraintGroup](#)
- [cstStartGenConstraintConstraintGroup](#)
- [cstGenConstraintConstraintGroup](#)
- [cstStartGenTechUnreferencedConstraint](#)
- [cstGenTechUnreferencedConstraint](#)
- [cstStartGenCellViewUnreferencedConstraint](#)
- [cstGenCellViewUnreferencedConstraint](#)
- [cstStopGen](#)

cstStartGenConstraintGroupToId

```
cstGenStateId  
cstStartGenConstraintGroupToId(  
    cstConstraintGroupId groupId  
);
```

Description

This function starts the generator for the member constraints and constraint groups of the specified constraint group.

Arguments

<i>groupId</i>	The ID of the constraintGroup, which is to own this constraint.

Return Value

<i>cstGenStateID</i>	Returns the generator state identifier to be used when generating constraint groups which contains the specified constraint.

cstGenConstraintGroupTold

Boolean

```
cstGenConstraintGroupToId(  
    cstGenStateId  stateId,  
    cstId          *Id  
);
```

Description

Generates all defined constraints in the constraint group.

Arguments

<i>stateId</i>	The generator state ID to be used in generating the specified constraints.
<i>*Id</i>	The ID of the constraint or constraint group.

Return Value

<i>boolean</i>	Returns <code>TRUE</code> if all the constraint values returned are valid and can be used. Returns <code>FALSE</code> when the generator is finished.

cstStartGenTechConstraintGroup

```
cstGenStateId  
cstStartGenTechConstraintGroup(  
    cstTechFileId  techfileId,  
    Boolean        localOnly  
);
```

Description

This function starts generating the constraint groups in the specified technology file.

Arguments

<i>techfileId</i>	The database identifier of the technology file.
<i>localOnly</i>	Looks for a constraint group in an ITDB graph if the database ID is a technology database.

Return Value

<i>cstGenStateId</i>	Returns the generator state identifier to be used when generating constraint groups in the technology file.

cstGenTechConstraintGroup

Boolean

```
cstGenTechConstraintGroup(  
    cstGenStateId    stateId,  
    cstConstraintGroupId *groupId  
);
```

Description

This function generates a constraint group in the specified technology file.

Arguments

<i>stateId</i>	The generator state identifier to be used when generating constraint groups.
<i>*groupId</i>	The ID of the constraint group.

Return Value

<i>Boolean</i>	Returns <code>TRUE</code> when the <code>groupId</code> is valid. Returns <code>FALSE</code> when the <code>groupId</code> is invalid.

cstStartGenCellViewConstraintGroup

```
cstGenStateId  
cstStartGenCellViewConstraintGroup(  
    dbCellViewId cellViewId  
);
```

Description

This function starts generating the constraint groups in the specified cellview.

Arguments

<i>cellViewId</i>	The database identifier of the cellview.

Return Value

<i>cstGenStateId</i>	Returns the generator state identifier to be used when generating constraint groups in the specified cellview.

cstGenCellViewConstraintGroup

Boolean

```
cstGenCellViewConstarintGroup(  
    cstGenStateId      stateId,  
    cstConstraintGroupId *groupId  
);
```

Description

This function generates a constraint group in the specified cellview.

Arguments

<i>stateId</i>	The generator state identifier to be used when generating constraint groups.
<i>*groupId</i>	The ID of the constraint group.

Return Value

<i>Boolean</i>	Returns <code>TRUE</code> when the groupId is valid. Returns <code>FALSE</code> when the groupId is invalid.

cstStartGenConstraintConstraintGroup

```
cstGenStateId  
cstStartGenConstraintConstraintGroup(  
    cstConstraintId constraintId  
);
```

Description

This function starts generating the constraint groups which contains the specified constraint.

Arguments

<i>constraintId</i>	The ID of the constraint.

Return Value

<i>cstGenStateId</i>	Returns the generator state identifier to be used when generating constraint groups in a constraint.

cstGenConstraintConstraintGroup

Boolean

```
cstGenConstraintConstarintGroup(  
    cstGenStateId      stateId,  
    cstConstraintGroupId *groupId  
);
```

Description

This function generates a constraint group which contains the specified constraint.

Arguments

<i>stateId</i>	The generator state identifier to be used when generating constraint groups.
<i>*groupId</i>	The ID of the constraint group.

Return Value

<i>Boolean</i>	Returns <code>TRUE</code> when the groupId is valid. Returns <code>FALSE</code> when the groupId is invalid.

cstStartGenTechUnreferencedConstraint

```
cstGenStateId  
cstStartGenTechUnreferencedConstraint(  
    cstTechFileId techfileId  
);
```

Description

This function starts generating all the unreferenced constraints not belonging to any constraint group in the specified technology file.

Arguments

<i>techfileId</i>	The database identifier of the technology file.

Return Value

<i>cstGenStateId</i>	Returns the generator state identifier to be used when generating unreferenced constraints in the technology file.

cstGenTechUnreferencedConstraint

Boolean

```
cstGenTechUnreferencedConstraint(  
    cstGenStateId    stateId,  
    cstConstraintId  *constraintId  
);
```

Description

This function generates all the unreferenced constraints not belonging to any constraint group in the specified technology file.

Arguments

<i>stateId</i>	The generator state identifier to be used when generating unreferenced constraints.
<i>*constraintId</i>	The identification of the constraint.

Return Value

<i>Boolean</i>	Returns <code>TRUE</code> when the groupId is valid. Returns <code>FALSE</code> when the groupId is invalid.

cstStartGenCellViewUnreferencedConstraint

```
cstGenStateId  
cstStartGenCellViewUnreferencedConstraint(  
    dbCellViewId cellViewId  
);
```

Description

This function starts generating all the unreferenced constraints not belonging to any constraint group in the specified cellview.

Arguments

<i>cellViewId</i>	The database id of the cellview.

Return Value

<i>cstGenStateId</i>	Returns the generator state identifier to be used when generating unreferenced constraints in the cellview.

cstGenCellViewUnreferencedConstraint

Boolean

```
cstGenCellViewUnreferencedConstraint(  
    cstGenStateId    stateId,  
    cstConstraintId  *constraintId  
);
```

Description

This function generates all the unreferenced constraints not belonging to any constraint group in the specified cellview.

Arguments

<i>stateId</i>	The generator state identifier to be used when generating unreferenced constraints.
<i>*constraintId</i>	The identification of the constraint.

Return Value

<i>Boolean</i>	Returns TRUE when the groupId is valid. Returns FALSE when the groupId is invalid.

cstStopGen

```
void  
cstStopGen(  
    cstGenStateId stateId  
);
```

Description

This function releases the space occupied by the generator pointed to the stateId.

Arguments

<i>stateId</i>	The generator state identifier for which the generation is to be stopped.

Return Value

<i>void</i>	NONE

20nm CST APIs

The new CST C APIs are used to traverse the hierarchy of the given constraint group. The traversal includes all constraint groups nested within the specified constraint group and all constraint groups reached through nested constraintGroupHeaders. The first constraint that matches the lookup criteria is returned. If no match is found, then a nullId is returned.

The following CST C APIs have been added for 20nm.

- [cstFindFirstLayerConstraint](#)
- [cstFindFirstLayerPairConstraint](#)
- [cstFindFirstLayerArrayConstraint](#)
- [cstFindFirstSimpleConstraint](#)
- [cstGet1DTableValue](#)
- [cstGet2DTableValue](#)
- [cstGetTwoWidthTableValue](#)

cstFindFirstLayerConstraint

```
cstConstraintID
cstFindFirstLayerConstraint(
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    cstLayerNum             layerNum,
    cstPurpose              purpose,
    dbPropType              paramListType,
    dbPropValue             listParams,
    Boolean                 onlyHard
);
```

Description

Returns the first layer constraint in the given constraint group which matches the given criteria.

Arguments

<i>groupId</i>	The ID of the constraintGroup in which the constraint is searched.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>layerNum</i>	The layer number of the constraint. If the layer number is unknown, an error results
<i>purpose</i>	The purpose number of the layer constraint.
<i>paramListType</i>	A database constant variable, which is dbcILLListType.
<i>listParams</i>	A SKILL list of constraint parameters. The default value is <code>ilcNil</code>. The input constraintParams is in a format of <code>list(list('paramKeyWord{int float string list}))...</code> For example, <code>list(list('ConnectivityType "sameNet") list('PGNet t))</code>
<i>onlyHard</i>	If this is true, then the first matching hard constraint is found, else the first matching constraint is found.

Cadence Integrators Toolkit Database Reference

CST Access C Functions

Return Value

cstConstraintID

Returns the *ID* of the matching constraint found in the group. If no matching constraint is found, a null ID is returned.

cstFindFirstLayerPairConstraint

```
cstConstraintId
cstFindFirstLayerPairConstraint (
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    cstLayerNum             layerNum1,
    cstPurpose              purpose1,
    cstLayerNum             layerNum2,
    cstPurpose              purpose2,
    dbPropType              paramListType,
    dbPropValue             listParams,
    Boolean                 onlyHard
);
```

Description

Returns the first layerPair constraint in the given constraint group which matches the given criteria.

Arguments

<i>groupId</i>	The ID of the constraintGroup in which the constraint is searched.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>layerNum1</i>	The layer number of the first layer-purpose pair constraint. If the layer number is unknown, an error results
<i>purpose1</i>	The purpose number of the first layer-purpose pair constraint.
<i>layerNum2</i>	The layer number of the second layer-purpose pair constraint, if a two-layer constraint is being updated.
<i>purpose2</i>	The purpose number of the second layer-purpose pair constraint.
<i>paramListType</i>	A database constant variable, which is dbcILListType.

Cadence Integrators Toolkit Database Reference

CST Access C Functions

listParams

A SKILL list of constraint parameters. The default value is `ilcNil`. The input `constraintParams` is in a format of `list(list('paramKeyWord{int | float | string | list}))...)`

For example, `list(list('ConnectivityType
 "sameNet")
 list('PGNet t))`

onlyHard

If this is true, then the first matching hard constraint is found, else the first matching constraint is found.

Return Value

cstConstraintId

Returns the *ID* of the matching constraint found in the group. If no matching constraint is found, a null ID is returned.

cstFindFirstLayerArrayConstraint

```
cstConstraintId
cstFindFirstLayerArrayConstraint (
    cstConstraintGroupId    groupId,
    cppConstString          constraintDefName,
    unsigned int            numLayers,
    cstLayerNum             layerNums,
    cstPurpose              purposes,
    dbPropType              paramListType,
    dbPropValue             listParams,
    Boolean                 onlyHard
);
```

Description

Returns the first layerArray constraint in the given constraint group which matches the given criteria.

Arguments

<i>groupId</i>	The ID of the constraintGroup in which the constraint is searched.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>numLayers</i>	The number of layers in the array of layerNums
<i>layerNums</i>	The layer number of the array constraint. An array contains layer numbers. If the layer number is unknown, an error results.
<i>purposes</i>	The purpose number of the layer array constraint. An array contains purpose numbers.
<i>paramListType</i>	A database constant variable, which is dbcILLListType.
<i>listParams</i>	A SKILL list of constraint parameters. The default value is ilcNil. The input constraintParams is in a format of list(list('paramKeyWord{int float string list}))...) For example, list(list('ConnectivityType "sameNet") list('PGNet t))

Cadence Integrators Toolkit Database Reference

CST Access C Functions

onlyHard

If this is true, then the first matching hard constraint is found, else the first matching constraint is found.

Return Value

cstConstraintId

Returns the *ID* of the matching constraint found in the group. If no matching constraint is found, a null ID is returned.

cstFindFirstSimpleConstraint

```
cstConstraintId  
cstFindFirstSimpleConstraint(  
    cstConstraintGroupId    groupId,  
    cppConstString          constraintDefName,  
    dbPropType              paramListType,  
    dbPropValue             listParams,  
    Boolean                  onlyHard  
);
```

Description

Returns the first simple constraint in the given constraint group which matches the given criteria.

Arguments

<i>groupId</i>	The ID of the constraintGroup in which the constraint is searched.
<i>constraintDefName</i>	The mapped DFII constraint name used to get a corresponding oaConstraintDef.
<i>paramListType</i>	A database constant variable, which is dbcILLListType.
<i>listParams</i>	A SKILL list of constraint parameters. The default value is ilcNil. The input constraintParams is in a format of list(list('paramKeyWord{int float string list})...) For example, list(list('ConnectivityType "sameNet") list('PGNet t))
<i>onlyHard</i>	If this is true, then the first matching hard constraint is found, else the first matching constraint is found.

Return Value

<i>cstConstraintId</i>	Returns the <i>ID</i> of the matching constraint found in the group. If no matching constraint is found, a null ID is returned.

cstGet1DTableValue

Boolean

```
cstGet1DTableValue(  
    cstConstraintId  constraintId,  
    dbPropType       indexType,  
    dbPropValue      indexValue,  
    dbPropType       &valueType,  
    dbPropValue      &value,  
);
```

Description

Returns the value of the 1DTable at the given index. The type of value and the value are set in the valueType and value arguments of the function.

Arguments

<i>constraintId</i>	This is the constraint ID.
<i>indexType</i>	The type of number, such as integer, float, etc.
<i>indexValue</i>	This is the value of the index.
<i>valueType</i>	The type of the value found.
<i>value</i>	This is the found value.

Return Value

<i>Boolean</i>	Returns <code>true</code> if successful.

cstGet2DTableValue

Boolean

```
cstGet2DTableValue(  
    cstConstraintId  constraintId,  
    dbPropType       index1Type,  
    dbPropValue      index1Value,  
    dbPropType       index2Type,  
    dbPropValue      index2Value,  
    dbPropType       &valueType,  
    dbPropValue      &value,  
);
```

Description

Returns the value of the 2DTable at the given indexes. The type of value and the value are set in the valueType and value arguments of the function.

Arguments

<i>constraintId</i>	This is the constraint ID.
<i>index1Type</i>	The type of number, such as integer, float, etc.
<i>index1Value</i>	This is the value of the index.
<i>index2Type</i>	The type of number, such as integer, float, etc.
<i>index2Value</i>	This is the value of the index.
<i>valueType</i>	The type of the value found.
<i>value</i>	This is the found value.

Return Value

<i>Boolean</i>	Returns <code>true</code> if successful.

cstGetTwoWidthTableValue

Boolean

```
cstGetTwoWidthTableValue(  
    cstConstraintId  constraintId,  
    dbPropType       index1Type,  
    dbPropValue      index1Value,  
    dbPropType       index2Type,  
    dbPropValue      index2Value,  
    dbPropType       index3Type,  
    dbPropValue      index3Value,  
    dbPropType       &valueType,  
    dbPropValue      &value,  
);
```

Description

Returns the value of the two widths table at the given indexes. The type of value and the value are set in the valueType and value arguments of the function.

Arguments

<i>constraintId</i>	This is the constraint ID.
<i>index1Type</i>	The type of number, such as integer, float, etc.
<i>index1Value</i>	This is the value of the index.
<i>index2Type</i>	The type of number, such as integer, float, etc.
<i>index2Value</i>	This is the value of the index.
<i>index3Type</i>	The type of number, such as integer, float, etc.
<i>index3Value</i>	This is the value of the index.
<i>valueType</i>	The type of the value found.
<i>value</i>	This is the found value.

Return Value

<i>Boolean</i>	Returns <code>true</code> if successful.

Cadence Integrators Toolkit Database Reference

CST Access C Functions

Cadence Integrators Toolkit Database Reference

CST Access C Functions

InPlace Cover Obstruction Database Access Functions

The InPlaceCoverObstruction C APIs are used to create or destroy InPlaceCoverObstruction objects and to retrieve or set InPlaceCoverObstruction attributes. This chapter describes the following InPlaceCoverObstruction database C APIs:

- [dbCreateInPlaceCoverObstruction](#)
- [dbDestroyInPlaceCoverObstruction](#)
- [dbGetInPlaceCoverObstructionAbstractFileName](#)
- [dbGetInPlaceCoverObstructionAbstractFilePath](#)
- [dbGetInPlaceCoverObstructionAbstractViewName](#)
- [dbGetInPlaceCoverObstructionBloat](#)
- [dbGetInPlaceCoverObstructionBlockageAttributeValue](#)
- [dbGetInPlaceCoverObstructionBlockageModel](#)
- [dbGetInPlaceCoverObstructionDoughnutHalo](#)
- [dbGetInPlaceCoverObstructionLayers](#)
- [dbGetInPlaceCoverObstructionMaxMask](#)
- [dbGetInPlaceCoverObstructionMergeDistance](#)
- [dbGetInPlaceCoverObstructionMinMask](#)
- [dbGetInPlaceCoverObstructionNeedsRemodeling](#)
- [dbGetInPlaceCoverObstructionPinRemodeling](#)
- [dbGetInPlaceCoverObstructionPinCutOut](#)
- [dbGetInPlaceCoverObstructionSpacingModel](#)

Cadence Integrators Toolkit Database Reference

InPlace Cover Obstruction Database Access Functions

- dbInPlaceCoverObstructionExists
- dbSetInPlaceCoverObstructionAbstractFileName
- dbSetInPlaceCoverObstructionAbstractFilePath
- dbSetInPlaceCoverObstructionAbstractViewName
- dbIsInPlaceCoverObstructionCurrent
- dbSetInPlaceCoverObstructionBloat
- dbSetInPlaceCoverObstructionBlockageAttributeValue
- dbSetInPlaceCoverObstructionBlockageModel
- dbSetInPlaceCoverObstructionCurrent
- dbSetInPlaceCoverObstructionDoughnutHalo
- dbSetInPlaceCoverObstructionMergeDistance
- dbSetInPlaceCoverObstructionNeedsRemodeling
- dbSetInPlaceCoverObstructionPinCutOut
- dbSetInPlaceCoverObstructionPinRemodeling
- dbSetInPlaceCoverObstructionSpacingModel

dbCreateInPlaceCoverObstruction

Boolean

```
dbCreateInPlaceCoverObstruction(  
    dbInstHeaderId instHeaderId,  
    dbIntRange *maskRange  
);
```

Description

Creates an InPlaceCoverObstruction object on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>*maskRange</i>	Integer mask range (minMask, maxMask) for the InPlaceCoverObstruction to be created on the instance header.

Return Value

<i>Boolean</i>	TRUE, if the InPlaceCoverObstruction object is created on the specified instance header. False, if the InPlaceCoverObstruction object is not created on the specified instance header.

Example

```
dbPoint          origin = {10, 20};  
  
dbInstId         inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId  instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange      range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)
```

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.

dbDestroyInPlaceCoverObstruction

Boolean

```
dbDestroyInPlaceCoverObstruction(  
    dbInstHeaderId instHeaderId  
);
```

Description

Destroys an InPlaceCoverObstruction object on the given instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.

Return Value

<i>Boolean</i>	TRUE, if the InPlaceCoverObstruction object on the given instance header is destroyed. FALSE, if the operation fails.

Example

```
dbPoint          origin = {10, 20};  
dbInstId         inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId  instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
  
if (dbInPlaceCoverObstructionExists((instHeaderId)) {  
    dbDestroyInPlaceCoverObstruction(instHeaderId)  
}
```

Destroys the InPlaceCoverObstruction object on the instance header *instHeaderId*.

dbGetInPlaceCoverObstructionAbstractFileName

String

```
dbGetInPlaceCoverObstructionAbstractFileName(  
    dbInstHeaderId  instHeaderId  
);
```

Description

Returns value of the InPlaceCoverObstruction AbstractFileName attribute for the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.

Return Value

<i>String</i>	Valid value of the InPlaceCoverObstruction AbstractFileName attribute.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbGetInPlaceCoverObstructionAbstractFileName(instHeaderId)
```

Returns the InPlaceCoverObstruction AbstractFileName attribute value for the instance header *instHeaderId*.

dbGetInPlaceCoverObstructionAbstractFilePath

String

```
dbGetInPlaceCoverObstructionAbstractFilePath(  
    dbInstHeaderId instHeaderId  
);
```

Description

Returns value of the InPlaceCoverObstruction AbstractFilePath attribute for the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.

Return Value

<i>String</i>	Value of the InPlaceCoverObstruction AbstractFilePath attribute for the specified instance header.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbGetInPlaceCoverObstructionAbstractFilePath(instHeaderId)
```

Returns the InPlaceCoverObstruction AbstractFilePath attribute value for the instance header *instHeaderId*.

dbGetInPlaceCoverObstructionAbstractViewName

```
String  
dbGetInPlaceCoverObstructionAbstractViewName(  
    dbInstHeaderId    instHeaderId  
);
```

Description

Returns value of the InPlaceCoverObstruction AbstractViewName attribute for the specified instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

String Value of the AbstractViewName attribute.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbGetInPlaceCoverObstructionAbstractViewName(instHeaderId)
```

Returns the InPlaceCoverObstruction AbstractViewName attribute value for the instance header *instHeaderId*.

dbGetInPlaceCoverObstructionBloat

```
int
dbGetInPlaceCoverObstructionBloat (
    dbInstHeaderId    instHeaderId,
    dbLayer            layerNum
);
```

Description

Returns value of the InPlaceCoverObstruction Bloat attribute from the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

<i>int</i>	Valid Bloat attribute values.
	If the operation fails, returns 0 and displays a warning message.

Example

```
dbPoint      origin = {10, 20};
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());
dbIntRange   range = {dbcClosedRangeType, 1, 4};

dbCreateInPlaceCoverObstruction(instHeaderId, &range)
dbLayer      *layersArray;
int          num;

dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)
if (num > 0) {
    dbGetInPlaceCoverObstructionBloat(instHeaderId, layersArray[0]) //
}
```

Returns the InPlaceCoverObstruction Bloat attribute value for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbGetInPlaceCoverObstructionBlockageAttributeValue

```
float  
dbGetInPlaceCoverObstructionBlockageAttributeValue(  
    dbInstHeaderId    instHeaderId,  
    dbLayer           layerNum  
);
```

Description

Returns value of the InPlaceCoverObstruction BlockageAttributeValue attribute from the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

<i>float</i>	Valid BlockageAttributeValue values. If the operation fails, returns 0 and displays a warning message.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
  
dbLayer      *layersArray;  
int          num;  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
  
if (num > 0) {  
    dbGetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId,  
        layersArray[0])  
}
```

Returns the InPlaceCoverObstruction BlockageAttributeValue attribute for the instance header *instHeaderId* on the layer *layerArray[0]*.

dbGetInPlaceCoverObstructionBlockageModel

```
dbBlockageModelType  
    dbGetInPlaceCoverObstructionBlockageModel(  
        dbInstHeaderId    instHeaderId,  
        dbLayer           layerNum  
    );
```

Description

Returns value of the InPlaceCoverObstruction BlockageModel attribute from the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

<i>dbBlockageModelType</i>	Valid BlockageModel value.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer      *layersArray;  
int          num;  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
  
if (num > 0) {  
    dbGetInPlaceCoverObstructionBlockageModel(instHeaderId, layersArray[0])  
    // returns a valid value of dbcFullCoverBlockageModel.  
}
```

Returns the InPlaceCoverObstruction BlockageModel attribute value for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbGetInPlaceCoverObstructionDoughnutHalo

```
int
dbGetInPlaceCoverObstructionDoughnutHalo (
    dbInstHeaderId    instHeaderId,
    dbLayer            layerNum
);
```

Description

Returns value of the InPlaceCoverObstruction DoughnutHalo attribute from the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

<i>int</i>	Valid DoughnutHalo values. If the operation fails, returns 0 and displays a warning message.

Example

```
dbPoint      origin = {10, 20};
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());
dbIntRange   range = {dbcClosedRangeType, 1, 4};

dbCreateInPlaceCoverObstruction(instHeaderId, &range)

dbLayer      *layersArray;
int          num;

dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)

if (num > 0) {
    dbGetInPlaceCoverObstructionDoughnutHalo(instHeaderId, layersArray[0])
}
```

Returns the InPlaceCoverObstruction DoughnutHalo attribute value for the instance header *instHeaderId* on the layer *layerArray[0]*.

dbGetInPlaceCoverObstructionLayers

Boolean

```
dbGetInPlaceCoverObstructionLayers (
    dbInstHeaderId    instHeaderId,
    int               *num,
    dbLayer           **layerNum
);
```

Description

Returns an array of physical layer numbers in the technology database, with the mask value lying between the `minMask` and `maxMask` values.

Arguments

<i>instHeaderId</i>	ID of the instance header on which the InPlaceCoverObstruction object is present.
<i>num</i>	Number of layers in the output dbLayer array.
<i>layerNum</i>	Array of physical layers with mask value in between the <code>minMask</code> and <code>maxMask</code> value specified at the time of creating the InPlaceCoverObstruction object.

Return Value

<i>Boolean</i>	TRUE, if the InPlaceCoverObstruction object is present on the instance header. FALSE, if the InPlaceCoverObstruction object is not present on the instance header.

Cadence Integrators Toolkit Database Reference

InPlace Cover Obstruction Database Access Functions

Example

```
dbPoint      origin = {10, 20};
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());
dbIntRange   range = {dbcClosedRangeType, 1, 4};

dbCreateInPlaceCoverObstruction(instHeaderId, &range)

dbLayer      *layersArray;
int          num;
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)
```

Returns a list of physical layer numbers 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.

dbGetInPlaceCoverObstructionMaxMask

```
int
dbGetInPlaceCoverObstructionMaxMask(
    dbInstHeaderId    instHeaderId
);
```

Description

Returns value of the InPlaceCoverObstruction MaxMask attribute from the specified instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

int Valid integer MaxMask values are returned in case of success.
If the operation fails, returns 0 and displays a warning message.

Example

```
dbPoint                      origin = {10, 20};

dbInstId                      inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());
dbIntRange                      range = {dbcClosedRangeType, 1, 4};

dbCreateInPlaceCoverObstruction(instHeaderId, &range)
dbGetInPlaceCoverObstructionMaxMask(instHeaderId)
// returns a valid value of 4
```

dbGetInPlaceCoverObstructionMergeDistance

```
float  
dbGetInPlaceCoverObstructionMergeDistance (  
    dbInstHeaderId    instHeaderId,  
    dbLayer           layerNum  
):
```

Description

Returns value of the InPlaceCoverObstruction MergeDistance attribute for the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

<i>float</i>	Valid MergeDistance value. If the operation fails, returns 0 and displays a warning message.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer *layersArray;  
int num;  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) {  
    dbGetInPlaceCoverObstructionMergeDistance(instHeaderId, layersArray[0])  
}
```

Returns the InPlaceCoverObstruction MergeDistance attribute value for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbGetInPlaceCoverObstructionMinMask

```
int
dbGetInPlaceCoverObstructionMinMask(
    dbInstHeaderId    instHeaderId
);
```

Description

Returns value of the InPlaceCoverObstruction MinMask attribute from the specified instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

int Valid MinMask values.
If the operation fails, returns 0 and displays a warning message.

Example

```
dbPoint          origin = {10, 20};
dbInstId         inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId   instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());

dbIntRange       range = {dbcClosedRangeType, 1, 4};

dbCreateInPlaceCoverObstruction(instHeaderId, &range)
dbGetInPlaceCoverObstructionMinMask(instHeaderId)
// returns a valid value of 1
```

dbGetInPlaceCoverObstructionNeedsRemodeling

Boolean

```
dbGetInPlaceCoverObstructionNeedsRemodeling(  
    dbInstHeaderId    instHeaderId  
);
```

Description

Returns value of the InPlaceCoverObstruction NeedsRemodeling attribute from the specified instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

Boolean Returns TRUE or FALSE.

Example

```
dbPoint                      origin = {10, 20};  
  
dbInstId                      inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange                      range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbGetInPlaceCoverObstructionNeedsRemodeling(instHeaderId)  
// returns a valid value of False
```

dbGetInPlaceCoverObstructionPinRemodeling

dbPinRemodelingType

```
dbGetInPlaceCoverObstructionPinRemodeling(  
    dbInstHeaderId    instHeaderId  
);
```

Description

Returns value of the InPlaceCoverObstruction PinRemodeling attribute from the specified instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

dbPinRemodelingType

Valid PinRemodeling attribute values.

Example

```
dbPoint                      origin = {10, 20};  
  
dbInstId                    inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange                range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbGetInPlaceCoverObstructionPinRemodeling(instHeaderId)  
// returns a valid value of dbcNonePinRemodeling.
```


dbGetInPlaceCoverObstructionPinCutOut

Boolean

```
dbGetInPlaceCoverObstructionPinCutOut (  
    dbInstHeaderId    instHeaderId,  
    dbLayer           layerNum  
);
```

Description

Returns value of the InPlaceCoverObstruction PinCutOut attribute for the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

<i>Boolean</i>	TRUE, if the PinCutOut attribute value for the specified instance is set to t. FALSE, if the PinCutOut attribute value for the specified instance is not set. This is also the default value of the PinCutOut attribute when the InPlaceCoverObstruction is created.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer *layersArray;  
int num;  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) {  
    dbGetInPlaceCoverObstructionPinCutOut(instHeaderId, layersArray[0])  
}
```

Cadence Integrators Toolkit Database Reference

InPlace Cover Obstruction Database Access Functions

Returns the InPlaceCoverObstruction PinCutOut attribute value for the instance header `instHeaderId` on the layer `layerArray[0]`.

dbGetInPlaceCoverObstructionSpacingModel

dbSpacingModelType

```
dbGetInPlaceCoverObstructionSpacingModel(  
    dbInstHeaderId  instHeaderId,  
    dbLayer         layerNum  
);
```

Description

Returns value of the InPlaceCoverObstruction SpacingModel attribute from the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.

Return Value

dbSpacingModelType

Valid SpacingModel values.

Example

```
dbPoint      origin = {10, 20};  
  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer      *layersArray;  
int          num;  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
  
if (num > 0) {  
    dbGetInPlaceCoverObstructionSpacingModel(instHeaderId, layersArray[0])  
    // returns a valid value of dbcSpacingModel  
}
```

dbInPlaceCoverObstructionExists

Boolean

```
dbInPlaceCoverObstructionExists(  
    dbInstHeaderId    instHeaderId  
);
```

Description

Checks whether an InPlaceCoverObstruction object is present on the specified instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

Boolean TRUE, if the InPlaceCoverObstruction object is present on the specified instance header.

FALSE, if the InPlaceCoverObstruction object is not present on the specified instance header.

Example

```
dbPoint                      origin = {10, 20};  
dbInstId                    inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange                  range = {dbcClosedRangeType, 1, 4};  
  
if (!dbInPlaceCoverObstructionExists((instHeaderId)) {  
    dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
}
```

Checks if an InPlaceCoverObstruction object is present on the instance header *instHeaderId*. An InPlaceCoverObstruction object is created if it is not present.

dbResetInPlaceCoverObstruction

Boolean

```
dbResetInPlaceCoverObstruction(  
    dbInstHeaderId          instHeaderId,  
    dbInPlaceCoverObstructionResetParamType resetType  
);
```

Description

Resets the values of the layer blockage parameters.

Arguments

instHeaderId ID of an instance header.

resetType An enumerated constant that specifies the reset type.

```
typedef enum dbInPlaceCoverObstructionResetParamType {  
    dbcInPlaceCoverObstructionUnknownResetParam = 0,  
    dbcInPlaceCoverObstructionBlockageModelResetParam = 1,  
    dbcInPlaceCoverObstructionBloatResetParam = 2,  
    dbcInPlaceCoverObstructionDoughnutHaloResetParam = 3,  
    dbcInPlaceCoverObstructionSpacingModelResetParam = 4,  
    dbcInPlaceCoverObstructionBlockageAttributeValueResetParam =  
    5,  
    dbcInPlaceCoverObstructionMergeDistanceResetParam = 6,  
    dbcInPlaceCoverObstructionPinCutOutResetParam = 7  
};
```

Return Value

Boolean TRUE, if the layer blockage type is reset to the specified reset value.

 FALSE, if the values of the layer blockage parameter are not reset.

dbSetInPlaceCoverObstructionAbstractFileName

Boolean

```
dbSetInPlaceCoverObstructionAbstractFileName(  
    dbInstHeaderId    instHeaderId,  
    constString       fileName  
)
```

Description

Sets value of the InPlaceCoverObstruction AbstractFileName attribute on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>fileName</i>	A valid AbstractFileName value.

Return Value

<i>Boolean</i>	TRUE, if the specified AbstractFileName attribute value is set for the specified instance header. FALSE, if the operation fails.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range);  
dbSetInPlaceCoverObstructionAbstractFileName(instHeaderId, "newFileName");
```

Sets the InPlaceCoverObstruction AbstractFileName attribute value of "newFileName" for the instance header *instHeaderId*.

dbSetInPlaceCoverObstructionAbstractFilePath

Boolean

```
dbSetInPlaceCoverObstructionAbstractFilePath(  
    dbInstHeaderId instHeaderId,  
    constString      filePath  
);
```

Description

Sets value of the InPlaceCoverObstruction AbstractFilePath attribute on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>filePath</i>	A valid AbstractFilePath value.

Return Value

<i>Boolean</i>	TRUE, if the specified AbstractFilePath attribute value is set for the specified instance header. FALSE, if the operation fails.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range);  
dbSetInPlaceCoverObstructionAbstractFilePath (instHeaderId, "newFilePath");
```

Sets the InPlaceCoverObstruction AbstractFilePath attribute value of "newFilePath" for the instance header *instHeaderId*.

dbSetInPlaceCoverObstructionAbstractViewName

Boolean

```
dbSetInPlaceCoverObstructionAbstractViewName (  
    dbInstHeaderId instHeaderId,  
    constString      viewName  
)
```

Description

Sets value of the InPlaceCoverObstruction AbstractViewName attribute on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>viewName</i>	A valid AbstractViewName value.

Return Value

<i>Boolean</i>	TRUE, if the specified AbstractViewName attribute value is set for the specified instance header. FALSE, if the operation fails.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range);  
dbSetInPlaceCoverObstructionAbstractViewName (instHeaderId, "newViewName");
```

Sets the dbSetInPlaceCoverObstruction AbstractViewName attribute value of "newViewName" for the instance header *instHeaderId*.

dbIsInPlaceCoverObstructionCurrent

Boolean

```
dbIsInPlaceCoverObstructionCurrent(  
    dbInstHeaderId    instHeaderId  
);
```

Description

Checks whether the timestamp for the InPlaceCoverObstruction object is in sync with the timestamp of the master design for the given instance header.

Arguments

instHeaderId ID of an instance header.

Return Value

Boolean TRUE, if the timestamps are in sync.
 FALSE, if the timestamps are not in sync

Example

```
dbPoint                      origin = {10, 20};  
dbInstId                    inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId((oaInst*)inst)->getHeader());  
dbIntRange                 range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
  
if (!dbIsInPlaceCoverObstructionCurrent(instHeaderId)) {  
    dbSetInPlaceCoverObstructionCurrent(instHeaderId)  
}
```

Checks if the timestamp for the InPlaceCoverObstruction object is current and makes it current if it is not.

dbSetInPlaceCoverObstructionBloat

Boolean

```
dbSetInPlaceCoverObstructionBloat(  
    dbInstHeaderId    instHeaderId,  
    dbLayer           layerNum,  
    int               value  
);
```

Description

Sets value of the InPlaceCoverObstruction Bloat attribute on the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.
<i>value</i>	A valid Bloat value.

Return Value

<i>Boolean</i>	TRUE, if the InPlaceCoverObstruction Bloat attribute is set on the specified instance header on the given layer. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer      *layersArray;  
int          num;  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) {  
    dbSetInPlaceCoverObstructionBloat(instHeaderId, layersArray[0], 10)  
}
```

Sets the InPlaceCoverObstruction Bloat attribute value of 10 for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbSetInPlaceCoverObstructionBlockageAttributeValue

Boolean

```
dbSetInPlaceCoverObstructionBlockageAttributeValue (
    dbInstHeaderId    instHeaderId,
    dbLayer           layerNum,
    float             value
);
```

Description

Sets value of the InPlaceCoverObstruction BlockageAttributeValue attribute on the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.
<i>value</i>	Value of BlockageAttributeValue value that needs to be set.

Return Value

<i>Boolean</i>	TRUE, if the specified BlockageAttributeValue is set on the given instance header.
	FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());
dbIntRange   range = {dbcClosedRangeType, 1, 4};

dbCreateInPlaceCoverObstruction(instHeaderId, &range)
dbLayer      *layersArray;
int          num;

dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)
if (num > 0) {
    dbSetInPlaceCoverObstructionBlockageAttributeValue(instHeaderId,
        layersArray[0], 10.0)
}
```

Sets value of the InPlaceCoverObstruction BlockageAttributeValue attribute to 10.0 for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbSetInPlaceCoverObstructionBlockageModel

Boolean

```
dbSetInPlaceCoverObstructionBlockageModel(  
    dbInstHeaderId      instHeaderId,  
    dbLayer              layerNum,  
    dbBlockageModelType value  
);
```

Description

Sets value of the InPlaceCoverObstruction BlockageModel attribute on the specified instance header on the specified layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.
<i>value</i>	Valid BlockageModel value.

Return Value

<i>Boolean</i>	TRUE, if the operation is successful. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer      *layersArray;  
int          num;  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) {  
    dbSetInPlaceCoverObstructionBlockageModel(instHeaderId, layersArray[0],  
        dbcDoughnutBlockageModel  
    )  
}
```

Sets value of the InPlaceCoverObstruction BlockageModel attribute to 10 for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbSetInPlaceCoverObstructionCurrent

Boolean

```
dbSetInPlaceCoverObstructionCurrent(  
    dbInstHeaderId    instHeaderId  
);
```

Description

Updates value of the timestamp for the InPlaceCoverObstruction object with the value of the master design timestamp for the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.

Return Value

<i>Boolean</i>	TRUE, if the operation is successful. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbSetInPlaceCoverObstructionCurrent(instHeaderId)
```

Makes the timestamp for the InPlaceCoverObstruction object current.

dbSetInPlaceCoverObstructionDoughnutHalo

Boolean

```
dbSetInPlaceCoverObstructionDoughnutHalo(  
    dbInstHeaderId    instHeaderId,  
    dbLayer           layerNum,  
    int               value  
);
```

Description

Sets value of the InPlaceCoverObstruction DoughnutHalo attribute on the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.
<i>value</i>	Valid DoughnutHalo value that needs to be set.

Return Value

<i>Boolean</i>	TRUE, if the operation is successful. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
  
dbLayer      *layersArray;  
int          num;  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) {  
    dbSetInPlaceCoverObstructionDoughnutHalo(instHeaderId, layersArray[0], 10)  
}
```

Sets value of the InPlaceCoverObstruction DoughnutHalo attribute to 10 for the instance header *instHeaderId* on the layer *layersArray[0]*.

dbSetInPlaceCoverObstructionMergeDistance

Boolean

```
dbSetInPlaceCoverObstructionMergeDistance (
    dbInstHeaderId    instHeaderId,
    dbLayer            layerNum,
    float              value
);
```

Description

Sets value of the InPlaceCoverObstruction MergeDistance attribute on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.
<i>value</i>	A valid MergeDistance value.

Return Value

<i>Boolean</i>	TRUE, if the specified MergeDistance attribute value is set for the specified instance header. FALSE, if the operation fails.

Example

```
dbPoint origin = {10, 20};
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());
dbIntRange range = {dbcClosedRangeType, 1, 4};
dbCreateInPlaceCoverObstruction(instHeaderId, &range)
dbLayer *layersArray;
int num;
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)
if (num > 0) {
    dbSetInPlaceCoverObstructionMergeDistance(instHeaderId, layersArray[0], 20.0)
}
```

Cadence Integrators Toolkit Database Reference

InPlace Cover Obstruction Database Access Functions

Sets value of the InPlaceCoverObstruction MergeDistance attribute to 20.0 for the instance header `instHeaderId` on the layer `layersArray[0]`.

dbSetInPlaceCoverObstructionNeedsRemodeling

Boolean

```
dbSetInPlaceCoverObstructionNeedsRemodeling(  
    dbInstHeaderId    instHeaderId,  
    Boolean           value  
);
```

Description

Sets value of the InPlaceCoverObstruction NeedsRemodeling attribute on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>value</i>	Valid NeedsRemodeling value.

Return Value

<i>Boolean</i>	TRUE, if the operation is successful. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbSetInPlaceCoverObstructionNeedsRemodeling(instHeaderId, true)
```

Sets the InPlaceCoverObstruction NeedsRemodeling attribute value to `true` for the instance header `instHeaderId`.

dbSetInPlaceCoverObstructionPinCutOut

Boolean

```
dbSetInPlaceCoverObstructionPinCutOut (  
    dbInstHeaderId    instHeaderId,  
    dbLayer           layerNum,  
    Boolean            value  
);
```

Description

Sets value of the InPlaceCoverObstruction PinCutOut attribute on the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Physical layer number present in the technology file.
<i>value</i>	A valid PinCutOut value.

Return Value

<i>Boolean</i>	TRUE, if the specified PinCutOut attribute value is set for the specified instance header. FALSE, if the operation fails.

Example

```
dbPoint origin = {10, 20};  
dbInstId inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange range = {dbcClosedRangeType, 1, 4};  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer *layersArray;  
int num;  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) {  
    dbSetInPlaceCoverObstructionPinCutOut(instHeaderId, layersArray[0], true)  
}
```

Cadence Integrators Toolkit Database Reference

InPlace Cover Obstruction Database Access Functions

Sets value of the InPlaceCoverObstruction PinCutOut attribute to `true` for the instance header `instHeaderId` on the layer `layersArray[0]`.

dbSetInPlaceCoverObstructionPinRemodeling

Boolean

```
dbSetInPlaceCoverObstructionPinRemodeling(  
    dbInstHeaderId      instHeaderId,  
    dbPinRemodelingType value  
);
```

Description

Sets value of the InPlaceCoverObstruction PinRemodeling attribute on the specified instance header.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>value</i>	Valid PinRemodeling value.

Return Value

<i>Boolean</i>	TRUE, if the operation is successful. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbSetInPlaceCoverObstructionPinRemodeling(instHeaderId,  
dbcPerimeterOnlyPinRemodeling)
```

Sets the InPlaceCoverObstruction PinRemodeling attribute value of `dbcPerimeterOnlyPinRemodeling` for the instance header `instHeaderId`.

dbSetInPlaceCoverObstructionSpacingModel

Boolean

```
dbSetInPlaceCoverObstructionSpacingModel(  
    dbInstHeaderId      instHeaderId,  
    dbLayer              layerNum,  
    dbSpacingModelType   value  
);
```

Description

Sets value of the InPlaceCoverObstruction SpacingModel attribute on the specified instance header on the given layer.

Arguments

<i>instHeaderId</i>	ID of an instance header.
<i>layerNum</i>	Name of a physical layer number specified in the technology file.
<i>value</i>	A valid SpacingModel value.

Return Value

<i>Boolean</i>	TRUE, if the InPlaceCoverObstruction SpacingModel attribute value is set on the specified instance header on the given layer. FALSE, if the operation fails.

Example

```
dbPoint      origin = {10, 20};  
dbInstId     inst = dbCreateInst(cv, master, "inst", origin, dbcR0, 1);  
dbInstHeaderId instHeaderId = dbiGetId(((oaInst*)inst)->getHeader());  
dbIntRange   range = {dbcClosedRangeType, 1, 4};  
  
dbCreateInPlaceCoverObstruction(instHeaderId, &range)  
dbLayer      *layersArray;  
int          num;  
  
dbGetInPlaceCoverObstructionLayers(instHeaderId, &num, &layersArray)  
if (num > 0) { dbSetInPlaceCoverObstructionSpacingModel(instHeaderId,  
layersArray[0], dbcAsIsSpacingModel)  
}
```

Cadence Integrators Toolkit Database Reference

InPlace Cover Obstruction Database Access Functions

Sets value of the InPlaceCoverObstruction SpacingModel attribute to dbcAsIsSpacingModel for the instance header `instHeaderId` on the layer `layersArray[0]`.

Multi-Patterning Technology Functions

Multi-patterning is a technique in which two or more mask processes are used to manufacture a design layer. Colors are used to represent the individual masks.

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](#).

This chapter describes the following multi-patterning technology (MPT) functions:

- [MPT Database Functions](#)
- [MPT Technology File Functions](#)

All of the functions detailed in this section are for advanced nodes use only.

MPT Database Functions

This section describes the following MPT database functions:

- [dbCellViewAreLayerShiftsValid](#)
- [dbCellViewClearLayerShifts](#)
- [dbCellViewHasLayerShifts](#)
- [dbCellViewUpdateLayerShifts](#)
- [dbInstGetLayerShifts](#)
- [dbInstHasLayerShifts](#)
- [dbInstSetLayerShifts](#)
- [dbInstClearLayerShifts](#)
- [dbIsShapeColoringAllowed](#)
- [dbIsShapeColored](#)
- [dbGetShapeColor](#)
- [dbIsShapeColorLocked](#)
- [dbIsShapeColorLocked](#)
- [dbIsShapeColorLocked](#)
- [dbSetShapeColorLocked](#)
- [dbIsBlockageColored](#)
- [dbIsBlockageColoringAllowed](#)
- [dbGetBlockageColor](#)
- [dbSetBlockageColor](#)
- [dbGetViaLayer1Control](#)
- [dbGetViaLayer2Control](#)
- [dbGetViaCutLayerControl](#)
- [dbGetViaLayerControl](#)
- [dbIsViaColorStateLocked](#)

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

- [dbIsViaColorStateLayerLocked](#)
- [dbSetViaLayer1Control](#)
- [dbSetViaLayer2Control](#)
- [dbSetViaCutLayerControl](#)
- [dbSetViaLayerControl](#)
- [dbSetViaColorStateLocked](#)
- [dbSetViaColorStateLayerLocked](#)
- [dbSetViaColorInfo](#)
- [dbIsTrackPatternColored](#)
- [dbIsTrackPatternColorAlternating](#)
- [dbIsTrackPatternColoringAllowed](#)
- [dbGetTrackPatternFirstTrackColor](#)
- [dbSetTrackPatternFirstTrackColor](#)
- [dbSetTrackPatternColorAlternating](#)
- [dbCreateSameMaskGroup](#)
- [dbCreateDiffMaskGroup](#)
- [dbColoredShapeQuery](#)
- [dbGetIntegrationColorModel](#)
- [dbGetColorModel](#)
- [dbSetIntegrationColorModel](#)
- [dbSetColorModel](#)
- [dbGetShapeColorLockType](#)
- [dbSetShapeColorLockType](#)
- [dbGetShapeColorLockType](#)
- [dbSetShapeColorLockType](#)

dbCellViewAreLayerShiftsValid

Boolean

```
dbCellViewAreLayerShiftsValid(  
    dbCellViewId cellViewId  
);
```

Description

Indicates whether the per-layer shift is valid on the instances for the specified cellview.

Arguments

<i>cellViewId</i>	Cellview ID.

Return Value

Boolean	Returns <code>true</code> if there is valid per-layer shift on the instances in the specified cellview. Returns <code>false</code> if there is no valid per-layer shift on the instances in the specified cellview or there is an error.

dbCellViewClearLayerShifts

Boolean

```
dbCellViewClearLayerShifts(  
    dbCellViewId cellViewId  
);
```

Description

Clears the per-layer shifting information from the specified cellview.

Arguments

<i>cellViewId</i>	Cellview ID.

Return Value

Boolean	Returns <code>true</code> if per-layer color shifting information is successfully removed from the specified cellview. Returns <code>false</code> if per-layer color shifting information is not removed from the specified cellview.

dbCellViewHasLayerShifts

Boolean

```
dbCellViewHasLayerShifts(  
    dbCellViewId cellViewId  
);
```

Description

Indicates whether any per-layer color shifts exist in the specified cellview.

Arguments

<i>cellViewId</i>	Cellview ID.

Return Value

Boolean	Returns <code>true</code> if per-layer color shifts exist in the specified cellview. Returns <code>false</code> if per-layer color shifts do not exist in the specified cellview or there is an error.

dbCellViewUpdateLayerShifts

Boolean

```
dbCellViewUpdateLayerShifts(  
    dbCellViewId cellViewId  
);
```

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, all per-layer shift information on the instances in the cellview is removed.

Note: You must set the layer shift information for a cellview before setting the layer shift information on instances in the cellview. To do this, use the following:

```
if (!dbCellViewHasLayerShifts(cv)) {  
    dbCellViewUpdateLayerShifts(cv);  
}
```

Arguments

<i>cellViewId</i>	Cellview ID.

Return Value

Boolean	Returns <code>true</code> if per-layer color specifications are successfully updated. Returns <code>false</code> if per-layer color specifications are not updated or there is an error.

dbInstGetLayerShifts

Boolean

```
dbInstGetLayerShifts(  
    dbInstId      instId,  
    unsigned int  *numLayerShifts,  
    dbLayerShift  **layerShifts  
);
```

Description

Returns a list of layer-shift pairs that are applied to the layers in the master of the specified instance. If no layer shifts are specified for the instance, the return value for *numLayerShifts* is 0 and the *layerShifts* pointer is NULL.

Note: Memory for *layerShifts* is allocated automatically using the `malloc` function and this memory must be released by calling the `free` function.

Arguments

<i>instId</i>	Instance ID.
<i>*numLayerShifts</i>	Output argument. Returns the size of <i>layerShifts</i> array.
<i>**layerShifts</i>	Output argument. Returns an array of layer-shift pairs that is set on the specified instance. Each layer-shift pair is represented as {<i>layer shift</i>} where <i>layer</i> is the layer number and <i>shift</i> is one of the following enums: ■ <code>dbcNoShift</code>: There is no color shifting on this layer. ■ <code>dbcShift1Shift</code>: Shift colors by one (mask1Color to mask2Color, mask2Color to mask3Color, mask3Color to mask1Color). ■ <code>dbcShift2Shift</code>: Shift colors by two (mask1Color to mask3Color, mask2Color to mask1Color, mask3Color to mask2Color). ■ <code>dbcFixedMask1Shift</code>: Shapes with mask1Color are not changed; shift mask2Color to mask3Color, and mask3Color to mask2Color. ■ <code>dbcFixedMask2Shift</code>: Shapes with mask2Color are not changed; shift mask1Color to mask3Color, and mask3Color to mask1Color.

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

- `dbcFixedMask3Shift`: Shapes with `mask3Color` are not changed; shift `mask1Color` to `mask2Color`, and `mask2Color` to `mask1Color`.

Return Value

Boolean Returns `true` if the function runs successfully.
 Returns `false` if there is an error.

Example

```
dbInstId      inst;                // for example, ID from dbCreateInst();
unsigned int   numLayers            = 0;    // initialize to 0
dbLayerShift *layerShiftPairs; = NULL;    // initialize to NULL pointer

dbInstGetLayerShifts(inst, &numLayers, &layerShiftPairs);
```

Returns the layer shifts that are set for instance `inst` as a list of `numLayers` layer-shift pairs.

dbInstHasLayerShifts

Boolean

```
dbInstHasLayerShifts(  
    dbInstId instId  
);
```

Description

Indicates whether the per-layer color shifts are specified on the instance.

Note: You must set the layer shift information for a cellview before setting the layer shift information on instances in the cellview. To do this, use the following:

```
if (!dbCellViewHasLayerShifts(cv)) {  
    dbCellViewUpdateLayerShifts(cv);  
}
```

Arguments

<i>instId</i>	Instance ID

Return Value

Boolean	Returns <code>true</code> if per-layer color shifts exist on the specified instance. Returns <code>false</code> if per-layer color shifts do not exist on the specified instance.

dbInstSetLayerShifts

Boolean

```
dbInstSetLayerShifts(  
    dbInstId      instId,  
    unsigned int  numLayerShifts,  
    dbLayerShifts *layerShifts  
);
```

Description

Sets the master layer shifting information specified in the given list of layer-shift pairs.

Note: You must initialize the layer shift information for a cellview before setting the layer shift information on the instances in the cellview. To do this, use the following:

```
if (!dbCellViewHasLayerShifts(cv)) {  
    dbCellViewUpdateLayerShifts(cv);  
}
```

Arguments

<i>instId</i>	Instance ID.
<i>numLayerShifts</i>	Specifies a size of the layerShifts array.
<i>*layerShifts</i>	Specifies a list of layer-shift pairs to be set on the specified instance. Each layer-shift pair is represented as <code>{layer shift}</code> where <i>layer</i> is the layer number and <i>shift</i> is one of the following enums: ■ <code>dbcNoShift</code>: There is no color shifting on this layer. ■ <code>dbcShift1Shift</code>: Shift colors by one (mask1Color to mask2Color, mask2Color to mask3Color, mask3Color to mask1Color). ■ <code>dbcShift2Shift</code>: Shift colors by two (mask1Color to mask3Color, mask2Color to mask1Color, mask3Color to mask2Color). ■ <code>dbcFixedMask1Shift</code>: Shapes with mask1Color are not changed; shift mask2Color to mask3Color, and mask3Color to mask2Color.

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

- `dbcFixedMask2Shift`: Shapes with `mask2Color` are not changed; shift `mask1Color` to `mask3Color`, and `mask3Color` to `mask1Color`.
- `dbcFixedMask3Shift`: Shapes with `mask3Color` are not changed; shift `mask1Color` to `mask2Color`, and `mask2Color` to `mask1Color`.

Return Value

`Boolean` Returns `true` if the master layer shifting information is successfully set.
Returns `false` if there is an error.

Example

```
dbInstId inst; // for example, inst ID from dbCreateInst();
dbLayer layers[3] = {20, 21, 22};
dbLayerShift shifts[3] = {{layers[0], dbcShift1Shift}, {layers[1],
dbcFixedMask1Shift}, {layers[2], dbcNoShift}};
dbInstSetLayerShifts(inst, 3, shifts);
```

Sets layer shifts for instance `inst` to `dbcShift1Shift`, `dbcFixedMask1Shift`, and `dbcNoShift` on layers 20, 21, and 22, respectively.

dbInstClearLayerShifts

Boolean

```
dbInstClearLayerShifts(  
    dbInstId instId  
);
```

Description

Removes all per-layer color shifting information from the specified instance.

Arguments

<i>instId</i>	Instance ID.

Return Value

Boolean	Returns <code>true</code> if the per-layer color shifting information is successfully removed from the specified instance. Returns <code>false</code> if the per-layer color shifting information is not removed from the specified instance.

dbIsShapeColoringAllowed

Boolean

```
dbIsShapeColoringAllowed(  
    dbShapeId shapeId  
);
```

Description

Indicates whether the specified shape can be colored. The following shapes can be colored—arcs, lines, donuts, ellipses, path segments, paths, rectangles, and polygons.

Arguments

<i>shapeId</i>	Shape ID.

Return Value

Boolean	Returns <code>true</code> if the shape can be colored. Returns <code>false</code> if the shape cannot be colored or there is an error.

dbIsShapeColored

Boolean

```
dbIsShapeColored(  
    dbShapeId shapeId  
);
```

Description

Indicates whether the specified shape is colored in color other than `grayColor`.

Arguments

<i>shapeId</i>	Shape ID.

Return Value

Boolean	Returns <code>true</code> if the shape is not colored in <code>grayColor</code> . Returns <code>false</code> if the shape is colored in <code>grayColor</code> or there is an error.

dbGetShapeColor

```
dbMaskColorType  
dbGetShapeColor(  
    dbShapeId shapeId  
);
```

Description

Returns the assigned color of the specified shape. By default, all newly created shapes have been assigned the color as `grayColor`. The assigned color of a shape can be different from the user color specification. The assigned color is based in the context of the cellview in which the shape is present and it is not the effective color of the shape in the context of the complete design.

Arguments

<i>shapeId</i>	ID of the shape whose color is to be returned.

Return Value

<code>dbMaskColorType</code>	Returns the color of the shape, which can be <code>grayColor</code> , <code>mask1Color</code> , <code>mask2Color</code> , or <code>mask3Color</code> .

dbIsShapeColorLocked

Boolean

```
dbIsShapeColorLocked(  
    dbShapeId shapeId  
);
```

Description

Indicates whether the assigned color for the specified shape is locked.

Arguments

<i>shapeId</i>	Shape ID.

Return Value

Boolean	Returns <code>true</code> if the coloring of the specified shape is locked. Returns <code>false</code> if the coloring of the specified shape is not locked or there is an error.

dbSetShapeColorLocked

Boolean

```
dbSetShapeColorLocked(  
    dbShapeId shapeId,  
    Boolean value  
);
```

Description

Changes the state of the assigned coloring of the shape. This function results in an error if you try to change the lock state of a non-colorable shape. It also returns error if you try to change the lock state of a shape whose assigned color is `grayColor`.

Arguments

<i>shapeId</i>	Shape ID.
<i>value</i>	Indicates whether a lock state is to be set.

Return Value

Boolean	Returns <code>true</code> if the function runs successfully. Returns <code>false</code> if there is an error.

dbIsBlockageColored

Boolean

```
dbIsBlockageColored(  
    dbBlockageId blockageId  
);
```

Description

Indicates whether the specified blockage is colored in color other than `grayColor`.

Arguments

<i>blockageId</i>	Blockage ID.

Return Value

Boolean	Returns <code>true</code> if the blockage is not colored in <code>grayColor</code> . Returns <code>false</code> if the blockage is colored in <code>grayColor</code> or there is an error.

dbIsBlockageColoringAllowed

Boolean

```
dbIsBlockageColoringAllowed(  
    dbBlockageId blockageId  
);
```

Description

Indicates whether the specified blockage can be colored. Only layer blockages can be colored.

Arguments

<i>blockageId</i>	Blockage ID.

Return Value

Boolean	Returns <code>true</code> if the blockage can be colored. Returns <code>false</code> if the blockage cannot be colored or there is an error.

dbGetBlockageColor

```
dbMaskColorType  
dbGetBlockageColor(  
    dbBlockageId blockageId  
);
```

Description

Returns the assigned color of the blockage. The default color type is `grayColor`.

Arguments

<i>blockageId</i>	Blockage ID.

Return Value

<code>maskColorType</code>	Returns the assigned color of the blockage, which can be <code>grayColor</code> , <code>mask1Color</code> , <code>mask2Color</code> , <code>mask3Color</code> , <code>blackColor</code> , or <code>multiColor</code> .

dbSetBlockageColor

Boolean

```
dbSetBlockageColor(  
    dbBlockageId    blockageId,  
    dbMaskColorType maskColorType  
);
```

Description

Assigns color to the blockage. If you try to assign color to the blockage that cannot be colored, it results in an error.

Arguments

<i>blockageId</i>	Blockage ID.
<i>maskColorType</i>	Color to be assigned to the blockage. Valid values: <code>grayColor</code> , <code>mask1Color</code> , <code>mask2Color</code> , <code>mask3Color</code> , <code>blackColor</code> , or <code>multiColor</code> .

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbGetViaLayer1Control

```
dbColorControlType  
dbGetViaLayer1Control(  
    dbViaId viaId  
);
```

Description

Returns the color control assigned to the `layer1` of the via. The default color control is `noShifts`.

Arguments

<i>viaId</i>	Via ID.

Return Value

dbColorControlType Returns the color control assigned the `layerv1` of the via.

Valid values: `dbcNoColorColorControl`,
`dbcNoShiftColorControl`, `dbcShift1ColorControl`,
`dbcShift2ColorControl`, `dbcShift3ColorControl`,
`dbcShift4ColorControl`, `dbcShift5ColorControl`,
`dbcShift6ColorControl`, or `dbcShift7ColorControl`.

dbGetViaLayer2Control

```
dbColorControlType  
dbGetViaLayer2Control(  
    dbViaId viaId  
);
```

Description

Returns the color control assigned to the `layer2` of the via. The default color control is `noShifts`.

Arguments

<i>viaId</i>	Via ID.

Return Value

dbColorControlType Returns the color control assigned the `layer2` of the via.

Valid values: `dbcNoColorColorControl`,
`dbcNoShiftColorControl`, `dbcShift1ColorControl`,
`dbcShift2ColorControl`, `dbcShift3ColorControl`,
`dbcShift4ColorControl`, `dbcShift5ColorControl`,
`dbcShift6ColorControl`, or `dbcShift7ColorControl`.

dbGetViaCutLayerControl

```
dbColorControlType  
dbGetViaCutLayerControl(  
    dbViaId viaId  
);
```

Description

Returns the color control assigned to the cut layer of the via. Default color control is noShifts.

Arguments

<i>viaId</i>	Via ID.

Return Value

dbColorControlType Returns the color control assigned to the cut layer of the via.

Valid values: dbcNoColorColorControl, dbcNoShiftColorControl, dbcShift1ColorControl, dbcShift2ColorControl, dbcShift3ColorControl, dbcShift4ColorControl, dbcShift5ColorControl, dbcShift6ColorControl, or dbcShift7ColorControl.

dbGetViaLayerControl

```
dbColorControlType  
dbGetViaLayerControl(  
    dbViaId      viaId,  
    dbViaLayerType layer  
);
```

Description

Returns the color control type for the specified layer in a via.

Arguments

<i>viaId</i>	Database ID of a via.
<i>layer</i>	Name of a valid via layer type. Valid values: dbcLayer1ViaLayer, dbcLayer2ViaLayer, dbcCutLayerViaLayer

Return Value

<i>dbColorControlType</i>	Returns the color control assigned for the specified layer in a via. Valid values: dbcNoColorColorControl, dbcNoShiftColorControl, dbcShift1ColorControl, dbcShift2ColorControl, dbcShift3ColorControl, dbcShift4ColorControl, dbcShift5ColorControl, dbcShift6ColorControl, or dbcShift7ColorControl

Example

```
bool    retVal = dbSetViaLayerControl(viaId, dbcLayer1ViaLayer,  
dbcShift4ColorControl);  
dbColorControlType t1 = dbGetViaLayerControl(viaId, dbcLayer1ViaLayer);  
bool    retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer,  
true);  
bool    isLockedLayer1 = dbIsViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer);  
retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer, true);  
bool    isLockedLayer2 = dbIsViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer);
```


Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

In this example, first the color control type `dbcShift4ColorControl` is set for the layer `dbcLayer1ViaLayer` in the via `viaID`.

Next, the color state for layers `dbcLayer1ViaLayer` and `dbcLayer2ViaLayer` in the via `viaID` is changed to locked. `dbIsViaColorStateLayerLocked` is used to ensure that the color state for the specified layers has been changed.

dbIsViaColorStateLocked

Boolean

```
dbIsViaColorStateLocked(  
    dbViaId viaId  
);
```

Description

Indicates whether the coloring on the via is locked. The default value is `nil`.

Arguments

<i>viaId</i>	Via ID.

Return Value

Boolean	Returns <code>true</code> if via coloring is locked. Returns <code>false</code> if via coloring is unlocked or there is an error.

dbIsViaColorStateLayerLocked

Boolean

```
dbIsViaColorStateLayerLocked(  
    dbViaId      viaId,  
    dbViaLayerType layer  
);
```

Description

Indicates whether the coloring for the specified individual layer in a via is locked.

Arguments

<i>viaId</i>	Database ID of a via.
<i>layer</i>	Name of a valid via layer type. Valid values: dbcLayer1ViaLayer, dbcLayer2ViaLayer, dbcCutLayerViaLayer

Return Value

Boolean	Returns TRUE if the coloring state for the specified individual layer in a via is locked. Returns FALSE the coloring state for the specified individual layer in a via is not locked or if there is an error.

Example

```
bool    retVal = dbSetViaLayerControl(viaId, dbcLayer1ViaLayer,  
dbShift4ColorControl);  
dbColorControlType t1 = dbGetViaLayerControl(viaId, dbcLayer1ViaLayer);  
bool    retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer,  
true);  
bool    isLockedLayer1 = dbIsViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer);  
retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer, true);  
bool    isLockedLayer2 = dbIsViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer);
```

In this example, first the color control type `dbcShift4ColorControl` is set for the layer `dbcLayer1ViaLayer` in the via `viaId`.

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

Next, the color state for layers `dbcLayer1ViaLayer` and `dbcLayer2ViaLayer` in the via `viaID` is changed to locked. `dbIsViaColorStateLayerLocked` is used to ensure that the color state for the specified layers has been changed.

dbSetViaLayer1Control

Boolean

```
dbSetViaLayer1Control(  
    dbViaId          viaId,  
    dbColorControlType colorControlType  
);
```

Description

Sets the color control for the layer1 shapes of the via master. If you try to assign color control to the layer without allowed colored masks or if via color state is locked, it results in an error.

Arguments

<i>viaId</i>	Via ID.
<i>colorControlType</i>	Color control to be assigned to the layer1 shapes of the via master. Valid values: <code>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and so on, up to <code>dbcShiftxColorControl</code> , where <i>x</i> 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>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and <code>dbcShift2ColorControl</code> .

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbSetViaLayer2Control

Boolean

```
dbSetViaLayer2Control(  
    dbViaId          viaId,  
    dbColorControlType colorControlType  
);
```

Description

Sets the color control for the layer2 shapes of the via master. If you try to assign color control to the layer without allowed colored masks or if the via color state is locked, it results in an error.

Arguments

<i>viaId</i>	Via ID.
<i>colorControlType</i>	Color control to be assigned to the layer2 shapes of the via master. Valid values: <code>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and so on, up to <code>dbcShiftxColorControl</code> , where <i>x</i> 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>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and <code>dbcShift2ColorControl</code> .

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbSetViaCutLayerControl

Boolean

```
dbSetViaCutLayerControl(  
    dbViaId          viaId,  
    dbColorControlType colorControlType  
);
```

Description

Sets the color control for the cut layer shapes of the via master. If you try to assign color control to the layer without allowed colored masks or if via color state is locked, it results in an error.

Arguments

<i>viaId</i>	Via ID.
<i>colorControlType</i>	Color control to be assigned to the cut layer shapes of the via master. Valid values: <code>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and so on, up to <code>dbcShiftxColorControl</code> , where <i>x</i> 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>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and <code>dbcShift2ColorControl</code> .

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbSetViaLayerControl

Boolean

```
dbSetViaLayerControl(  
    dbViaId          viaId,  
    dbViaLayerType   layer,  
    dbColorControlType control  
);
```

Description

Sets the specified color control type for the specified layer in a via.

Arguments

<i>viaId</i>	Database ID of a via.
<i>layer</i>	Name of a valid via layer type. Valid values: <code>dbcLayer1ViaLayer</code> , <code>dbcLayer2ViaLayer</code> , <code>dbcCutLayerViaLayer</code>
<i>control</i>	Color control to be assigned to the specified layer in a via. Valid values: <code>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and so on, up to <code>dbcShiftxColorControl</code> , 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: <code>dbcNoColorColorControl</code> , <code>dbcNoShiftColorControl</code> , <code>dbcShift1ColorControl</code> , and <code>dbcShift2ColorControl</code> .

Return Value

Boolean	TRUE, if the specified color control type is set for the specified layer in the given via. FALSE, if the specified color control type is not set for the specified layer in the given via, or if the operation failed.

Example

```
bool    retVal = dbSetViaLayerControl(viaId, dbcLayer1ViaLayer,  
dbcShift4ColorControl);
```


Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

```
dbColorControlType t1 = dbGetViaLayerControl(viaId, dbcLayer1ViaLayer);
bool    retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer,
true);
bool    isLockedLayer1 = dbIsViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer);
retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer, true);
bool    isLockedLayer2 = dbIsViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer);
```

In this example, first the color control type `dbcShift4ColorControl` is set for the layer `dbcLayer1ViaLayer` in the via `viaID`.

Next, the color state for layers `dbcLayer1ViaLayer` and `dbcLayer2ViaLayer` in the via `viaID` is changed to locked. `dbIsViaColorStateLayerLocked` is used to ensure that the color state for the specified layers has been changed.

dbSetViaColorStateLocked

Boolean

```
dbSetViaColorStateLocked(  
    dbViaId viaId  
);
```

Description

Sets the color state for the specified via. If you try to set the color or color state of a via when the specified layer associated with the via is unbound or does not support coloring, it results in an error.

Arguments

<i>viaId</i>	Via ID.

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbSetViaColorStateLayerLocked

Boolean

```
dbSetViaColorStateLayerLocked(  
    dbViaId      viaId,  
    dbViaLayerType layer,  
    Boolean      value  
);
```

Description

Changes the color state for individual layers in a via.

Arguments

<i>viaId</i>	Database ID of a via.
<i>layer</i>	Value of a valid via layer type. Valid values: dbcLayer1ViaLayer, dbcLayer2ViaLayer, dbcCutLayerViaLayer
<i>value</i>	Specifies one of the two Boolean values: TRUE or FALSE. TRUE to change the color state to locked and FALSE to change the state to unlocked.

Return Value

Boolean	TRUE if the color state for a specific layer has been successfully set to the desired value. FALSE in case of an error.

Example

```
bool    retVal = dbSetViaLayerControl(viaId, dbcLayer1ViaLayer,  
dbShift4ColorControl);  
dbColorControlType t1 = dbGetViaLayerControl(viaId, dbcLayer1ViaLayer);  
bool    retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer,  
true);  
bool    isLockedLayer1 = dbIsViaColorStateLayerLocked(viaId, dbcLayer1ViaLayer);  
retValLocked = dbSetViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer, true);  
bool    isLockedLayer2 = dbIsViaColorStateLayerLocked(viaId, dbcLayer2ViaLayer);
```

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

In this example, first the color control type `dbcShift4ColorControl` is set for the layer `dbcLayer1ViaLayer` in the via `viaID`.

Next, the color state for layers `dbcLayer1ViaLayer` and `dbcLayer2ViaLayer` in the via `viaID` is changed to locked. `dbIsViaColorStateLayerLocked` is used to ensure that the color state for the specified layers has been changed.

dbSetViaColorInfo

Boolean

```
dbSetViaColorInfo(  
    dbViaId          viaId,  
    dbViaLayerColorInfo *layer1ControlInfo,  
    dbViaLayerColorInfo *layer2ControlInfo,  
    dbViaLayerColorInfo *cutLayerControlInfo  
);
```

Description

Updates the color controls and states of the given via. The color information arguments correspond to the three via layers (via layer1, via layer2, and cut layer).

A warning message is displayed with a return value of `FALSE` for error conditions such as the following:

- The color information argument is not set to `NULL` 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 `TRUE`).

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

Arguments

<i>viaId</i>	Database ID of the via.
<i>*layer1ColorInfo</i>	Specifies the color information for via layer1 as a C struct with two fields: ■ <i>control</i> Color control to be assigned to the via layer of the given via. Valid values: <code>dbcNoColorColorControl</code>, <code>dbcNoShiftColorControl</code>, <code>dbcShift1ColorControl</code>, and so on, up to <code>dbcShiftxColorControl</code>, where <i>x</i> 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: <code>dbcNoColorColorControl</code>, <code>dbcNoShiftColorControl</code>, <code>dbcShift1ColorControl</code>, and <code>dbcShift2ColorControl</code>. ■ <i>lock</i> Specifies one of two Boolean values: <code>TRUE</code> or <code>FALSE</code>, where <code>TRUE</code> changes the via layer state to locked, and <code>FALSE</code> changes the state to unlocked. Use <code>NULL</code> for no change to the via layer.
<i>*layer2ColorInfo</i>	Specifies the color information for via layer2, in the same format as <i>*layer1ColorInfo</i> .
<i>*cutLayerColorInfo</i>	Specifies the color information for the via cut layer, in the same format as <i>*layer1ColorInfo</i> .

Return Value

Boolean	Returns <code>TRUE</code> if the function runs successfully. Returns <code>FALSE</code> if there is an error.

dbIsTrackPatternColored

Boolean

```
dbIsTrackPatternColored(  
    dbTrackPatternId trackPatternId  
);
```

Description

Indicates whether the specified first track of the specified track pattern is colored in color other than `grayColor`.

Arguments

<i>trackPatternId</i>	ID for the track pattern.

Return Value

Boolean	Returns <code>true</code> if the first track of the track pattern is not colored in <code>grayColor</code> . Returns <code>false</code> if the first track of the track pattern is colored in <code>grayColor</code> or there is an error.

dbIsTrackPatternColorAlternating

Boolean

```
dbIsTrackPatternColorAlternating(  
    dbTrackPatternId trackPatternId  
);
```

Description

Indicates whether coloring for the specified track pattern is alternating.

Arguments

<i>trackPatternId</i>	ID for the track pattern.

Return Value

Boolean	Returns <code>true</code> if the track pattern coloring is alternating. Returns <code>false</code> if the track pattern coloring is not alternating or if there is an error.

dbIsTrackPatternColoringAllowed

Boolean

```
dbIsTrackPatternColoringAllowed(  
    dbTrackPatternId trackPatternId  
);
```

Description

Indicates whether the specified track pattern can be colored. Only track patterns where a routing layer is specified can be colored.

Arguments

<i>trackPatternId</i>	ID for the track pattern.

Return Value

Boolean	Returns <code>true</code> if the track pattern can be colored. Returns <code>false</code> if the track pattern cannot be colored or if there is an error.

dbGetTrackPatternFirstTrackColor

```
dbMaskColorType  
dbGetTrackPatternFirstTrackColor(  
    dbTrackPatternId trackPatternId  
);
```

Description

Returns the assigned color of the first track of the specified track pattern. The default color type is `grayColor`.

Arguments

<i>trackPatternId</i>	ID for the track pattern.

Return Value

<code>maskColorType</code>	Returns the assigned color of the first track of the specified track pattern, which can be <code>grayColor</code> , <code>mask1Color</code> , <code>mask2Color</code> , or <code>mask3Color</code> .

dbSetTrackPatternFirstTrackColor

Boolean

```
dbSetTrackPatternFirstTrackColor(  
    dbTrackPatternId trackPatternId,  
    dbMaskColor      maskColor  
);
```

Description

Assigns the color to the first track of the specified track pattern. If you assign color to the first track of the track pattern that cannot be colored, it results in an error.

Arguments

<i>trackPatternId</i>	ID for the track pattern.
<i>maskColor</i>	Color to be assigned to the first track of the specified track pattern. Valid values: <code>grayColor</code> , <code>mask1Color</code> , <code>mask2Color</code> , or <code>mask3Color</code> .

Return Value

Boolean	Returns <code>true</code> if the function runs successfully. Returns <code>false</code> if there is an error.

dbSetTrackPatternColorAlternating

Boolean

```
dbSetTrackPatternColorAlternating(  
    dbTrackPatternId trackPatternId,  
    Boolean           setLocked  
);
```

Description

Sets the coloring type of the specified track pattern. An error occurs if you try to set the color or color control of a track pattern when the track pattern has no routing layer or the routing layer is unbound and does not support coloring.

Arguments

<i>trackPatternId</i>	ID for the track pattern.
<i>setLocked</i>	Coloring type to be set.

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbCreateSameMaskGroup

```
dbGroupId  
dbCreateSameMaskGroup(  
    dbCellViewId cellViewId,  
    constString groupName  
);
```

Description

Creates same mask color group in the specified cellview.

Arguments

<i>cellViewId</i>	Cellview ID.
<i>groupName</i>	Name of the color group to be created.

Return Value

<i>groupId</i>	Returns the created same mask color group ID.

dbCreateDiffMaskGroup

```
dbGroupId  
dbCreateDiffMaskGroup(  
    dbCellViewId cellViewId,  
    constString  groupName,  
    dbGroupId    sameMaskGroup1,  
    dbGroupId    sameMaskGroup2  
);
```

Description

Creates diff mask color group in the specified cellview. This group defines a pair of groups of the shapes of different colors.

Arguments

<i>cellViewId</i>	Cellview ID.
<i>groupName</i>	Name of the color group to be created.
<i>sameMaskGroup1</i>	First group from the pair of groups of the shapes of different colors.
<i>sameMaskGroup2</i>	Second group from the pair of groups of the shapes of different colors.

Return Value

<i>groupId</i>	Returns the created diff mask color group ID.

dbColoredShapeQuery

```
void
dbColoredShapeQuery(
    dbCellViewId      cellViewId,
    dbLayer            layer,
    dbPurpose          purpose,
    dbBBox             *region,
    dbTransform        transform,
    unsigned int       filterSize,
    unsigned int       startLevel,
    unsigned int       stopLevel,
    dbShapeQueryConsumer shapeConsumer,
    void               *clientData
);
```

Description

Returns a list of pairs of all the shapes and their colors in a cellview whose bounding boxes overlap the region specified by *l_bBox*.

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.

Arguments

<i>cellViewId</i>	Cellview ID.
<i>layer</i>	Layer number or layer ID.
<i>purpose</i>	Purpose of the layer.
<i>*region</i>	Specifies the region that should overlap bounding boxes of all returned shapes.
<i>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.
<i>filterSize</i>	Shapes filter size. The query produces only shapes with a size larger than the filter size. If no <i>filterSize</i> argument is specified, the blockage filter size defaults to 0.

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

<i>startLevel</i>	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.
<i>stopLevel</i>	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.
<i>shapeConsumer</i>	Calls the <code>dbShapeQueryConsumer</code> function that returns the color of the shape using the corresponding <code>dbQueryStateId</code> .

Return Value

<i>void</i>	No return value.

dbGetIntegrationColorModel

Boolean

```
dbGetIntegrationColorModel(  
    dbCellViewId          cellViewId  
    dbIntegrationColorModelType *type  
);  
  
typedef enum dbIntegrationColorModelType {  
    dbcUnknownIntegrationColorModel=0,  
    dbcAnyIntegrationColorModel=1,  
    dbcLockedIntegrationColorModel=2  
} dbIntegrationColorModelType;
```

Description

Returns the *integration color model* constraint value for the specified cell view. The valid values are any or locked.

Arguments

<i>cellViewId</i>	Cellview ID.
* <i>type</i>	Integration color model type. Valid values: any or locked.

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if the constraint value does not exist for the cellview. In this case, the <code>type</code> field is left blank.

dbGetColorModel

Boolean

```
dbGetColorModel(  
    dbCellViewId      cellViewId,  
    dbColorModelType *type  
);
```

Description

Returns the value of *color model constraint* for the specified cell view. Possible values are *any*, *layerBased*, or *locked*. If the constraint value is not specified on the cellview, this function returns *false*.

Arguments

<i>cellViewId</i>	Cellview ID.
<i>*type</i>	Color model constraint type. Valid values: <i>any</i> , <i>layerBased</i> , or <i>locked</i> .

Return Value

Boolean	Returns <i>true</i> if function runs successfully. Returns <i>false</i> if the constraint value does not exist for the specified cellview. In this case, the <i>type</i> field is left blank.

dbSetIntegrationColorModel

```
void  
dbSetIntegrationColorModel(  
    dbCellViewId          cellViewId,  
    dbIntegrationColorModelType type  
);
```

Description

Sets the value of *integration color model* constraint for the specified cellview. Possible values are any, or locked.

Arguments

<i>cellViewId</i>	Cellview ID.
<i>type</i>	Integration color model to be set. Valid values: any, or locked.

Return Value

void	No return value.

dbSetColorModel

Boolean

```
dbSetColorModel(  
    dbCellViewId    cellViewId,  
    dbColorModelType type  
);
```

```
typedef enum dbColorModelType {  
    dbcAnyColorModel=0,  
    dbcLayerBasedColorModel=1,  
    dbcLockedColorModel=2  
} dbColorModelType;
```

Description

Sets the value of *color model* constraint for the specified cell view. Possible values are *any*, *layerBased*, or *locked*.

Arguments

<i>cellViewId</i>	Cellview ID.
<i>type</i>	Color model to be set. Valid values: <i>any</i> , <i>layerBased</i> , or <i>locked</i> .

Return Value

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

dbGetShapeColorLockType

```
dbColorLockType  
dbGetShapeColorLockType(  
    dbShapeId shapeId  
);
```

Description

Returns the color lock type assigned to the specified shape. If the shape is not locked, `unlocked` is returned. By default, `dbcUser` lock type is assigned to locked shapes.

Arguments

<i>shapeId</i>	The database ID of a shape.

Return Value

<i>dbColorLockType</i>	Color lock type assigned to the specified shape. Possible values are <code>dbcUnlocked</code> , <code>dbcUser</code> , and <code>dbcSystem</code> . ■ <code>dbcUser</code>: Coloring of the shape is locked. ■ <code>dbcSystem</code>: Coloring of the shape was locked by automated coloring tools. ■ <code>dbcUnlocked</code>: The shape is not locked.

Example

```
dbShapeId shape = dbCreateRect(...);  
dbSetShapeColor(shape "mask1Color");  
assert(dbGetShapeColorLockType(shape) == dbcUnlocked);  
dbSetShapeColorLocked(shape TRUE);  
assert(dbGetShapeColorLockType(shape) == dbcUser);  
dbSetShapeColorLockType(shape dbcSystem);  
assert(dbGetShapeColorLockType(shape) == dbcSystem);  
dbSetShapeColorLockType(shape dbcUnlocked);  
assert(dbGetShapeColorLockType(shape) == dbcUnlocked);  
assert(!dbIsShapeColorLocked(shape));
```

dbSetShapeColorLockType

Boolean

```
dbSetShapeColorLockType(  
    dbShapeId      shapeId,  
    dbColorLockType lockType  
);
```

Description

Sets the color lock type for the specified shape.

Arguments

shapeId

The database ID of a shape.

lockType

Color lock type to assign to the specified shape.

Valid input values are integer constants `dbcUnlocked`, `dbcUser`, `dbcSystem`, respectively.

- `dbcUser`
Sets coloring of the via layer to locked, with lock type `dbcUser`. All via layers locked in the older versions of Virtuoso have these lock type.
- `dbcSystem`
Sets coloring of the via layer to locked, with lock type `dbcSystem`. This lock type indicates that the via layer was locked by an automated coloring tool.
- `dbcUnlocked`
Unlocks coloring of the via layer and clears information about its lock type.
- Default value for existing locked shapes: `dbcUser`

Default value for existing locked shapes: `dbcUser`

Return Value

Boolean

Returns `true` if the specified lock type is successfully set.

Returns `false` if the lock type is not set.

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

Example

```
dbShapeId shape = dbCreateRect(...);

dbSetShapeColor(shape "mask1Color");
assert(dbGetShapeColorLockType(shape) == dbcUnlocked);
dbSetShapeColorLocked(shape TRUE);
assert(dbGetShapeColorLockType(shape) == dbcUser);
dbSetShapeColorLockType(shape dbcSystem);
assert(dbGetShapeColorLockType(shape) == dbcSystem);
dbSetShapeColorLockType(shape dbcUnlocked);
assert(dbGetShapeColorLockType(shape) == dbcUnlocked);
assert(!dbIsShapeColorLocked(shape));
```

dbGetViaLayerColorLockType

```
dbColorLockType  
dbGetViaLayerColorLockType(  
    dbViaId      viaId,  
    dbViaLayerType layer  
);
```

Description

Returns the color lock type assigned to the specified via layer.

Arguments

<i>viaId</i>	The database ID of a via layer.
<i>viaLayerType</i>	Specifies the via layer type. Valid values: layer1, layer2, cutLayer

Return Value

<i>dbColorLockType</i>	Color lock type assigned to the specified via layer. Color lock type assigned to the specified shape. Possible values are dbcUnlocked, dbcUser, and dbcSystem. ■ dbcUser: Coloring of the shape is locked. ■ dbcSystem: Coloring of the shape was locked by automated coloring tools. ■ dbcUnlocked: The shape is not locked.

Example

```
dbViaId via = dbCreateStdVia(...);  
  
dbViaLayerColorInfo l1 = { dbcShift1ColorControl, FALSE };  
dbSetViaColorInfo(via l1 NULL NULL);  
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcUnlocked);  
dbSetViaColorStateLayerLocked(via dbcLayer1ViaLayer TRUE);
```


Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

```
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcUser);  
dbSetViaLayerColorLockType(via dbcLayer1ViaLayer dbcSystem);  
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcSystem);  
dbSetViaLayerColorLockType(via dbcLayer1ViaLayer dbcUnlocked);  
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcUnlocked);  
assert(!dbIsViaColorStateLayerLocked(via dbcLayer1ViaLayer));
```

dbSetViaLayerColorLockType

Boolean

```
dbSetViaLayerColorLockType(  
    dbViaId      viaId,  
    dbViaLayerType layer,  
    dbColorLockType value  
);
```

Description

Sets the color lock type for the specified via layer.

Arguments

<i>viaId</i>	The database ID of a via layer.
<i>layer</i>	Specifies the via layer type. Valid values: <code>layer1</code> , <code>layer2</code> , <code>cutLayer</code>
<i>value</i>	Color lock type to assign to the specified via layer. Valid input values are integer constants <code>dbcUnlocked</code> , <code>dbcUser</code> , <code>dbcSystem</code> , respectively. ■ <code>dbcUser</code> Sets coloring of the via layer to locked, with lock type <code>dbcUser</code>. All via layers locked in the older versions of Virtuoso have these lock type. ■ <code>dbcSystem</code> Sets coloring of the via layer to locked, with lock type <code>dbcSystem</code>. This lock type indicates that the via layer was locked by an automated coloring tool. ■ <code>dbcUnlocked</code> Unlocks coloring of the via layer and clears information about its lock type.

Default value for existing locked shapes: `dbcUser`

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

Return Value

Boolean If `true` the specified lock type is successfully set.
 If `false` the lock type is not set.

Example

```
dbViaId via = dbCreateStdVia(...);

dbViaLayerColorInfo l1 = { dbcShift1ColorControl, FALSE };
dbSetViaColorInfo(via l1 NULL NULL);
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcUnlocked);
dbSetViaColorStateLayerLocked(via dbcLayer1ViaLayer TRUE);
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcUser);
dbSetViaLayerColorLockType(via dbcLayer1ViaLayer dbcSystem);
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcSystem);
dbSetViaLayerColorLockType(via dbcLayer1ViaLayer dbcUnlocked);
assert(dbGetViaLayerLockType(via dbcLayer1ViaLayer) == dbcUnlocked);
assert(!dbIsViaColorStateLayerLocked(via dbcLayer1ViaLayer));
```

MPT Technology File Functions

This section describes the following MPT technology file functions:

- [techGetIntegrationColorModel](#) on page 1821
- [techGetLayerNumColorMasks](#) on page 1822
- [techGetStdViaDefCutColoring](#) on page 1823
- [techGetTechCutColoring](#) on page 1824
- [techIsStdViaDefCutColoringSet](#) on page 1825
- [techSetIntegrationColorModel](#) on page 1826
- [techSetLayerNumColorMasks](#) on page 1827
- [techSetStdViaDefCutColoring](#) on page 1828
- [techSetTechCutColoring](#) on page 1829

techGetIntegrationColorModel

Boolean

```
techGetIntegrationColorModel(  
    techFileId                techfileId,  
    techIntegrationColorModelType *type  
);  
  
typedef enum techIntegrationColorModelType {  
    techcAnyIntegrationColorModel=0,  
    techcLockedIntegrationColorModel=1  
} techIntegrationColorModelType;
```

Description

Returns the value of *integration color model* constraint for the specified technology file. Possible values are *any* or *locked*.

Arguments

<i>techId</i>	The identifier of the technology database.
<i>*type</i>	Integration color model type. Valid values: <i>any</i> or <i>locked</i> .

Return Values

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if the constraint value does not exist for the cellview. In this case, the <code>type</code> field is left blank.

techGetLayerNumColorMasks

```
unsigned int  
techGetLayerNumColorMasks(  
    techFileId  techId,  
    techLayerNum layerNum  
);
```

Description

Returns the number of allowed color masks on the specified layer. The return value 0 indicates coloring is not supported on the specified layer.

Arguments

<i>techId</i>	The identifier of the technology database.
<i>layerNum</i>	The layer name or layer number.

Return Values

unsigned int	Returns the number of allowed color masks on the layer (0, 2, or 3).

techGetStdViaDefCutColoring

```
techCutColoringType  
techGetStdViaDefCutColoring(  
    techViaDefId viaDefId  
);
```

Description

Returns the cut coloring pattern used for the cut patterns created for the specified `StdVia`.

Arguments

<i>viaDefId</i>	The database identifier of the via definition.

Return Values

<code>techCutColoringType</code>	Returns the cut coloring pattern on the specified <code>StdViadef</code> .

techGetTechCutColoring

```
techCutColoringType  
techGetTechCutColoring(  
    techfileId techfile  
);
```

Description

Returns the default cut coloring pattern used for the cut patterns created for all `StdVias` defined in the technology. This function returns the *checkerboard* pattern by default if the cut pattern coloring is not explicitly specified.

Arguments

<i>techfile</i>	The identifier of the technology database.

Return Values

<code>techCutColoringType</code>	Returns the default cut coloring model for <code>stdViaDef</code> cuts in the specified technology.

techIsStdViaDefCutColoringSet

Boolean

```
techIsStdViaDefCutColoringSet(  
    techViaDefId viaDefId  
);
```

Description

Indicates whether the cut coloring pattern has been explicitly set on the specified StdViaDef.

Arguments

<i>viaDefId</i>	The database identifier of the via definition.

Return Values

Boolean	Returns <code>true</code> if the cut coloring pattern is explicitly set on the StdViaDef. Returns <code>false</code> if the cut coloring pattern is not set or if there is an error.

techSetIntegrationColorModel

```
void  
techSetIntegrationColorModel(  
    techFileId          techfileId,  
    techIntegrationColorModelType type  
);  
  
typedef enum techIntegrationColorModelType {  
    techcAnyIntegrationColorModel=0,  
    techcLockedIntegrationColorModel=1  
} techIntegrationColorModelType;
```

Description

Sets the value of the *integration color model* constraint for the specified technology file. Possible values are *any* or *locked*.

Arguments

<i>techfileId</i>	The identifier of the technology database.
<i>type</i>	Integration color model type. Valid values: <i>any</i> or <i>locked</i> .

Return Values

<code>void</code>	No return value.

techSetLayerNumColorMasks

Boolean

```
techSetLayerNumColorMasks(  
    techFileId    techfileId,  
    techLayerNum layerNum,  
    unsigned int  numColorMasks  
);
```

Description

Assigns the number of allowed color masks on the specified layer. The valid color masks that can be set are 0, 2, and 3. If you try to set the color mask number to a value 1 or greater than 3, the function results in an error. This function does not set the attribute on the DFII system reserved layers.

Arguments

<i>techfileId</i>	The identifier of the technology database.
<i>layerNum</i>	Layer name or layer number.
<i>numColorMasks</i>	Number of color masks to be set Valid Values: 0, 2, and 3.

Return Values

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

techSetStdViaDefCutColoring

Boolean

```
techSetStdViaDefCutColoring(  
    techViaDefId      viaDefId,  
    techCutColoringType type  
);
```

Description

Sets the cut coloring pattern to be used for the cut patterns created for the specified `StdVia`.

Arguments

<i>viaDefId</i>	The database identifier of the via definition.

Return Values

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

techSetTechCutColoring

Boolean

```
techSetTechCutColoring(  
    techfileId      techfile,  
    techCutColoringType type  
);
```

Description

Sets the default cut coloring pattern to be used for the cut patterns created for all `StdVias` defined in the specified technology file.

Arguments

<i>techfileId</i>	The identifier of the technology database.
<i>type</i>	Cut coloring pattern to be set. Valid Values: <code>checkerboard</code> or <code>alternatingRows</code> .

Return Values

Boolean	Returns <code>true</code> if function runs successfully. Returns <code>false</code> if there is an error.

Cadence Integrators Toolkit Database Reference

Multi-Patterning Technology Functions

CDBA Photonics Functions

This chapter includes the following topics:

- [Photonics Functions](#)
- [Curvy Core Functions](#)
- [Waveguide Functions](#)

You need the `Virtuoso_Photonics_Option` license with `Virtuoso_MultiTech_Framework` (95022) to use the functionality discussed in this section. For more information, see *License Checkout for Virtuoso Photonics Platform* in [License Checkout for Virtuoso Photonics Platform](#) of *Virtuoso Software Licensing and Configuration User Guide*.

Photonics Functions

This section covers the photonics functions:

- [dbGetPhotonicPinFigAngle](#)
- [dbGetPhotonicPinFigRadius](#)
- [dbGetPhotonicPinFigWidth](#)
- [dbIsPhotonicsEnabled](#)
- [dbCellViewHasPhotonicPinFig](#)
- [dbSetPhotonicPinFigAngle](#)
- [dbSetPhotonicPinFigRadius](#)
- [dbSetPhotonicPinFigWidth](#)
- [dbUnsetPhotonicPinFigAngle](#)
- [dbUnsetPhotonicPinFigRadius](#)
- [dbUnsetPhotonicPinFigWidth](#)

Note: You need the `Virtuoso_Photonics_Option` license to use the functionality discussed in this section. For information about obtaining the required license, contact your local Cadence representative.

dbGetPhotonicPinFigAngle

Boolean

```
dbGetPhotonicPinFigAngle(  
    dbFigId    figId,  
    float      *value  
);
```

Description

Returns the attribute value of the angle for the specified photonics pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.
<i>value</i>	Attribute value of the angle.

Return Value

<i>Boolean</i>	TRUE if the attribute value of the angle for the specified photonics pinFig is returned. FALSE if the pinFig is not a photonic pin or the attribute is not set.

Example

```
dbGetPhotonicPinFigAngle(figId, rtnValuePtr)  
=> true
```

The attribute value of the angle for the photonic pinFig *figId* is retrieved.

dbGetPhotonicPinFigRadius

Boolean

```
dbGetPhotonicPinFigRadius(  
    dbFigId    figId,  
    int        *value  
);
```

Description

Returns the attribute value of the radius for the specified photonics pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.
<i>value</i>	Attribute value of the radius.

Return Value

<i>Boolean</i>	TRUE if the attribute value of the radius for the specified photonics pinFig is returned. FALSE if the pinFig is not a photonic pin or the attribute is not set.

Example

```
dbGetPhotonicPinFigRadius(figId, rtnValuePtr)  
=> false
```

The pinFig *figId* is not a photonic pinFig or the radius attribute is not set.

dbGetPhotonicPinFigWidth

Boolean

```
dbGetPhotonicPinFigWidth(  
    dbFigId      figId,  
    unsigned int *value  
);
```

Description

Returns the attribute value of the width for the specified photonics pinFig.

Arguments

<i>d_pinFigId</i>	Database ID of a photonics pinFig.
<i>value</i>	Attribute value of the angle.

Return Value

Boolean	TRUE if the attribute value of the width for the specified photonics pinFig is returned. FALSE if the pinFig is not a photonic pin or the attribute is not set.

Example

```
dbGetPhotonicPinFigWidth(figId, rtnValuePtr)  
=> true
```

The attribute value of the width for the photonic pinFig *figId* is retrieved.

dbIsPhotonicsEnabled

```
dbIsPhotonicsEnabled(  
    );
```

Description

Confirms whether the photonics-related functionality is enabled.

Note: Optical sigTypes cannot be set if photonics-related functionality is not enabled.

Arguments

None

Return Value

t	Photonics-related functionality is enabled.
nil	Photonics-related functionality is not enabled.

dbCellViewHasPhotonicPinFig

Boolean

```
dbCellViewHasPhotonicPinFig(  
    dbCellViewId cvId  
);
```

Description

Checks whether the specified cellview has a photonic pinFig.

Arguments

<i>cvId</i>	Database ID of a cellview.

Return Value

<i>Boolean</i>	TRUE if the specified cellview has a photonic pinFig. FALSE if the specified cellview does not have a photonic pinFig.

Example

```
dbCellViewHasPhotonicPinFig(cvId)  
=> true
```

The cellview *cvId* contains a photonic pinFig.

dbSetPhotonicPinFigAngle

Boolean

```
dbSetPhotonicPinFigAngle(  
    dbFigId    figId,  
    float      value  
);
```

Description

Sets the given attribute value as the angle of the photonic for the specified pinFig.

Arguments

<i>figId</i>	Database ID of a photonics pinFig.
<i>value</i>	Attribute value of angle.

Return Value

<i>Boolean</i>	TRUE if the specified attribute value is set as the angle of the photonics pinFig. FALSE if the angle attribute value is not set.

Example

```
dbSetPhotonicPinFigAngle(figId, value)  
=> true
```

The attribute value of the angle for the photonic pinFig *figId* is set to *value*.

dbSetPhotonicPinFigRadius

Boolean

```
dbSetPhotonicPinFigRadius(  
    dbFigId    figId,  
    int        value  
);
```

Description

Sets the given attribute value as the radius of the photonic for the specified pinFig.

Arguments

<i>figId</i>	Database ID of a photonics pinFig.
<i>value</i>	Attribute value of radius.

Return Value

<i>Boolean</i>	TRUE if the specified attribute value is set as the radius of the photonics pinFig. FALSE if the radius attribute value is not set.

Example

```
dbSetPhotonicPinFigRadius(figId, value)  
=> false
```

The attribute value of the radius for the pinFig *figId* is not set. This may imply that *figId* is not a photonic pinFig or the cellview that contains *figId* is not a layout cellview type.

dbSetPhotonicPinFigWidth

Boolean

```
dbSetPhotonicPinFigWidth(  
    dbFigId    figId,  
    unsigned int value  
);
```

Description

Sets the given attribute value as the width of the photonic for the specified pinFig.

Arguments

<i>figId</i>	Database ID of a photonics pinFig.
<i>value</i>	Attribute value of width.

Return Value

<i>Boolean</i>	TRUE if the specified attribute value is set as the width of the photonics pinFig. FALSE if the width attribute value is not set.

Example

```
dbSetPhotonicPinFigWidth(figId, value)  
=> true
```

The attribute value of the width for the specific pinFig *figId* is set to *value*.

dbUnsetPhotonicPinFigAngle

Boolean

```
dbUnsetPhotonicPinFigAngle(  
    dbFigId  figId  
);
```

Description

Resets the attribute value of the angle for the specified photonics pinFig.

Arguments

<code>figId</code>	Database ID of a photonics pinFig.

Return Value

<i>Boolean</i>	TRUE if the angle attribute value of the specified photonic pinFig is reset. FALSE if the angle attribute value is not reset.

Example

```
dbUnsetPhotonicPinFigAngle(figId)  
=> true
```

The attribute value of the angle for the photonic pinFig `figId` is reset.

dbUnsetPhotonicPinFigRadius

Boolean

```
dbUnsetPhotonicPinFigRadius(  
    dbFigId figId  
);
```

Description

Resets the attribute value of the radius for the specified photonics pinFig.

Arguments

<i>figId</i>	Database ID of a photonics pinFig.

Return Value

<i>Boolean</i>	TRUE if the radius attribute value of the specified photonic pinFig is reset. FALSE if the radius attribute value is not reset.

Example

```
dbUnsetPhotonicPinFigRadius(figId)  
=> false
```

The attribute value of the radius for the pinFig *figId* is not reset. This implies that *figId* is not a photonic pinFig.

dbUnsetPhotonicPinFigWidth

Boolean

```
dbUnsetPhotonicPinFigWidth(  
    dbFigId  figId  
);
```

Description

Resets the attribute value of the width for the specified photonics pinFig.

Arguments

<i>figId</i>	Database ID of a photonics pinFig

Return Value

<i>Boolean</i>	TRUE if the width attribute value of the specified photonic pinFig is reset. FALSE if the width attribute value is not reset.

Example

```
dbUnsetPhotonicPinFigWidth(figId)  
=> true
```

The attribute value of the width for the photonic pinFig *figId* is reset.

Curvy Core Functions

Curvy shapes are building blocks of photonic devices such as ring resonators, waveguide bends, and Bragg gratings. You can use these C functions to get information related to existing CurvyCore objects.

This section describes the following curvy core functions.

■ Functions to Check Curvy Core Objects

- ❑ cclsBoundary
- ❑ cclsCurve
- ❑ cclsFacet
- ❑ cclsLine
- ❑ cclsMath
- ❑ cclsPath
- ❑ cclsPolyCurve
- ❑ cclsPolyCurveSegment
- ❑ cclsShape
- ❑ cclsSurface

■ Functions to query object attributes

- ❑ ccGetCurveSpec
- ❑ ccGetFacetSpec
- ❑ ccGetFromFig
- ❑ ccGetPathBeginFacet
- ❑ ccGetPathBoundary
- ❑ ccGetPathCenterLine
- ❑ ccGetPathEndFacet
- ❑ ccGetPolyCurve
- ❑ ccGetSurfaceBoundary

■ Generator functions

Cadence Integrators Toolkit Database Reference

CDBA Photonics Functions

- ☐ [ccGenOAShape](#)
- ☐ [ccGenPolyCurveSegment](#)
- ☐ [ccGenSurfaceHole](#)
- ☐ [ccStartGenOAShape](#)
- ☐ [ccStartGenPolyCurveSegment](#)
- ☐ [ccStartGenSurfaceHole](#)
- ☐ [ccStopGen](#)

■ [Miscellaneous Functions](#)

- ☐ [ccFreePathWidths](#)
- ☐ [ccReleaseObject](#)

Functions to Check Curvy Core Objects

ccIsBoundary

Boolean

```
ccIsBoundary(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccBoundary object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccBoundary. FALSE if the specified object is not a ccBoundary object, or if an error occurred.

ccIsCurve

Boolean

```
ccIsCurve(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccCurve object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccCurve. FALSE if the specified object is not a ccCurve object, or if an error occurred.

ccIsFacet

Boolean

```
ccIsFacet(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccFacet object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccFacet object. FALSE if the specified object is not a ccFacet object, or if an error occurred.

ccIsLine

Boolean

```
ccIsLine(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccLine object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccLine. FALSE if the specified object is not a ccLine object, or if an error occurred.

Cadence Integrators Toolkit Database Reference

CDBA Photonics Functions

cclsMath

Boolean

```
ccIsMath(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a mathematical object, such as ccCurve, ccFacet, ccPolyCurve, or ccBoundary.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a mathematical object. FALSE if the specified object is not a mathematical object, or if an error occurred.

ccIsPath

Boolean

```
ccIsPath(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccPath object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccPath object. FALSE if the specified object is not a ccPath object, or if an error occurred.

ccIsPolyCurve

Boolean

```
ccIsPolyCurve(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccPolyCurve object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccPolyCurve object. FALSE if the specified object is not a ccPolyCurve object, or if an error occurred.

ccIsPolyCurveSegment

Boolean

```
ccIsPolyCurveSegment(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccPolyCurve segment object, such as ccCurve or ccFacet.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccPolyCurve segment object. FALSE if the specified object is not a ccPolyCurve segment object, or if an error occurred.

ccIsShape

Boolean

```
ccIsShape(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccShape object, such as ccPath, ccLine, or ccSurface.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccShape object. FALSE if the specified object is not a ccShape object, or if an error occurred.

ccIsSurface

Boolean

```
ccIsSurface(  
    ccId id  
);
```

Description

(ICADVM20.1 Photonics Only) Checks whether the specified CurvyCore object is a ccSurface object.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

<i>Boolean</i>	TRUE if the specified object is a ccSurface object. FALSE if the specified object is not a ccSurface object, or if an error occurred.

Functions to query object attributes

ccGetCurveSpec

ccCurveSpec

```
ccGetCurveSpec(  
    ccCurveId curveId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the curve specification for the specified ccCurve ID.

Arguments

<i>curveId</i>	ID of a ccCurve object.

Return Value

<i>ccCurveSpec</i>	A ccCurveSpec object with curve data.

The function allocates memory for the strings in the specifications in format (*.x*, *.y*).

Note: Use `free()` to free the strings after the procedure is complete.

Returns a STRUCT filled with zeros and no memory is allocated.

ccGetFacetSpec

ccFacetSpec

```
ccGetFacetSpec(  
    ccFacetId facetId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the facet specifications from the given ccFacet ID.

Arguments

<i>facetId</i>	ID of a ccFacet object.
	Note: An error occurs if <i>ccID</i> of some other type is specified.

Return Value

<i>ccFacetSpec</i>	A ccFacetSpec object filled with facet data.
	The function allocates memory for the strings in the specifications in format <code>(.origin.x,.origin.y)</code> .
	Note: Use <code>free()</code> to free the strings after the procedure.
	Returns a STRUCT filled with zeros and no memory is allocated.

ccGetFromFig

```
ccShapeId  
  
ccGetFromFig(  
    dbShapeId  shapeId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the CurvyCore shapes for the specified database shape.

Note: This operation increments the reference-count of the underlying *ccShapeHandle* object by 1.

Arguments

<i>curveId</i>	ID of an OpenAccess shape.
	Note: Type validation for the specified <i>shapeId</i> is not done.

Return Value

<i>ccShapeId</i>	ID of a <i>ccShape</i> object.
	Returns <i>ccNullId</i> in case of error.

ccGetPathBeginFacet

```
ccFacetId  
  
ccGetPathBeginFacet(  
    ccPathId pathId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the beginning facet of the specified ccPath object.

Arguments

<i>pathId</i>	ID of a ccPath object.
	Note: An error occurs if <i>ccId</i> of some other type is specified.

Return Value

<i>ccFacetId</i>	ID of a ccFacet object.
	Returns <code>ccNullId</code> in case of error.

ccGetPathBoundary

ccBoundaryId

```
ccGetPathBoundary(  
    ccPathId pathId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the boundary of the specified ccPath as a ccBoundary.

Arguments

<i>pathId</i>	ID of a ccPath.
	Note: An error occurs if <i>ccID</i> of some other type is specified.

Return Value

<i>ccBoundaryId</i>	A ccBoundary ID.
	Returns <code>ccNullId</code> in case of error.

ccGetPathCenterLine

ccPolyCurveId

```
ccGetPathCenterLine(  
    ccPathId pathId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the centerLine of the specified ccPath as an open polyCurve.

Arguments

pathId

ID of a ccPath object.

Note: An error occurs if *ccID* of some other type is specified.

Return Value

ccPolyCurveId

ID of a ccPolyCurve object.

Returns `ccNullId` in case of error.

ccGetPathWidths

ccPathWidthSpec

```
ccGetPathWidths(  
    ccPathId pathId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the width specifications for the specified ccPath.

Arguments

<i>pathId</i>	ID of a ccPath object.
	An error occurs if <i>ccID</i> of some other type is specified.

Return Value

<i>ccPathWidthSpec</i>	A ccPathWidthSpec object with width data. This function allocates memory for the strings in the width specification. Note: Use <u>ccFreePathWidths</u> to free the strings after the procedure is complete. Returns a STRUCT filled with zeros and no memory is allocated.

ccGetPathEndFacet

```
ccFacetId  
  
ccGetPathEndFacet(  
    ccPathId pathId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the ending facet of the specified ccPath object.

Arguments

<i>pathId</i>	ID of a ccPath object.
	Note: An error occurs if <i>ccID</i> of some other type is specified.

Return Value

<i>ccFacetId</i>	ID of a ccFacet object.
	Returns <code>ccNullId</code> in case of error.

ccGetPolyCurve

```
ccPolyCurveId  
  
ccGetPolyCurve(  
    ccLineId lineId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the mathematical polyCurve of the given ccLine.

Arguments

<i>pathId</i>	ID of a ccLine object.
	Note: An error occurs if <i>ccID</i> of some other type is specified.

Return Value

<i>ccPolyCurveId</i>	A ccPolyCurve ID.
	Returns <code>ccNullId</code> in case of error.

ccGetSurfaceBoundary

ccBoundaryId

```
ccGetSurfaceBoundary(  
    ccSurfaceId surfaceId  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the boundary of the specified ccSurface as a ccBoundary.

Arguments

<i>surfaceId</i>	ID of a ccSurface object. Note: An error occurs if <i>ccID</i> of some other type is specified.

Return Value

<i>ccBoundaryId</i>	A ccBoundary ID. Returns <code>ccNullId</code> in case of error.

Generator functions

ccGenOAShape

Boolean

```
ccGenOAShape (
    ccGenState  *pstate,
    dbShapeId   *id
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the ID of the next OpenAccess shape from the specified `ccGenState` created by ccStartGenOAShape.

Arguments

<i>pState</i>	State ID for the generator.
<i>id</i>	ID of the next OpenAccess shape produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> is returned if the operation succeeded and the generator advanced to next value. <code>FALSE</code> is returned if the operation succeeded but the generator reached the end of the list, or if an error occurred.

ccGenPolyCurveSegment

Boolean

```
ccGenPolyCurveSegment(  
    ccGenState  *pstate,  
    ccMathId    *id,  
    ccTransform *xform  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the ID of the next polyCurve segment and related ccTransform (in case of Facet segment) from the specified ccGenState created by ccStartGenPolyCurveSegment.

Arguments

pstate

State ID for the generator.

id

Pointer to the memory location where the ID of the next polyCurve segment produced by the generator will be written.

This value is undefined if FALSE is returned.

xform

Pointer to the memory location where the ccTransform of the next polyCurve segment produced by the generator will be written.

Note: ccTransform is retrieved only in case of the ccFacet segment. If the segment is a ccCurve, the value of *xform* remains unchanged.

Return Value

Boolean

TRUE is returned if the operation succeeded and the generator advanced to next value.

FALSE is returned if the operation succeeded but the generator reached the end of the list, or if an error occurred.

ccGenSurfaceHole

Boolean

```
ccGenSurfaceHole(  
    ccGenState      *pstate,  
    ccBoundaryId    *id  
);
```

Description

(ICADVM20.1 Photonics Only) Retrieves the ID of the next *ccBoundaryId* (hole) from the specified *ccGenState* created by ccStartGenSurfaceHole.

Arguments

pstate

State ID for the generator.

id

Pointer to the memory location where ID of the next *ccBoundaryId* produced by the generator will be written.

This value is undefined if `FALSE` is returned.

Return Value

Boolean

`TRUE` is returned if the operation succeeded and the generator advanced to next value.

`FALSE` is returned if the operation succeeded but the generator reached the end of the list, or if an error occurred.

ccStartGenOAShape

ccGenState

```
ccStartGenOAShape(  
    ccShapeId  shapeId  
);
```

Description

(ICADVM20.1 Photonics Only) Creates and initializes a generator for all OpenAccess shapes with the OpenAccess geometry of the specified shape. A generator state is returned, which is used with the [ccGenOAShape](#).

Note: Call the [ccStopGen](#) function at the end of generation loop to free the generator state.

Arguments

shapeId ID of a ccShape object.

Note: An error occurs if *ccID* of some other type is specified.

Return Value

ccGenState

State ID for the generator, which is:

```
.type = ccIdGenTypeOAShape,  
.status = ccGenStatusActive
```

In case of error, the following is returned:

```
.type = ccIdGenTypeNone,  
.status = ccGenStatusNull.
```

ccStartGenPolyCurveSegment

ccGenState

```
ccStartGenPolyCurveSegment(  
    ccMathId  pcId  
);
```

Description

(ICADVM20.1 Photonics Only) Creates and initializes a generator for all the segments of the specified *ccPolyCurve* or *ccBoundary*. A generator state is returned, which is used with the *ccGenPolyCurveSegment*.

Note: Call the *ccStopGen* function at the end of generation loop to free the generator state.

Arguments

pcId

ID of a *ccPolyCurve* or *ccBoundary* object.

Note: An error occurs if *ccID* of some other type is specified.

Return Value

ccGenState

State ID for the generator, which is:

```
.type = ccIdGenTypePolyCurveSegment,  
.status = ccGenStatusActive
```

In case of error, the following is returned:

```
.type = ccIdGenTypeNone,  
.status = ccGenStatusNull.
```

ccStartGenSurfaceHole

ccGenState

```
ccStartGenSurfaceHole(  
    ccSurfaceId  surfaceId  
);
```

Description

(ICADVM20.1 Photonics Only) Creates and initializes a generator for all the holes of the specified *ccSurface*. A generator state is returned, which is used with the [ccGenSurfaceHole](#).

Note: Call the [ccStopGen](#) function at the end of generation loop to free the generator state.

Arguments

surfaceId

ID of a *ccSurface* object.

Note: An error occurs if *ccID* of some other type is specified.

Return Value

ccGenState

State ID for the generator, which is:

```
.type = ccIdGenTypeSurfaceHole,  
.status = ccGenStatusActive
```

In case of error, the following is returned:

```
.type = ccIdGenTypeNone,  
.status = ccGenStatusNull
```

ccStopGen

void

```
ccStopGen(  
    ccGenState  *pstate  
);
```

Description

(ICADVM20.1 Photonics Only) Clears the specified `ccGenState` data. At the end of a generator loop, you must call the `ccStopGen` function to end generation. This frees the generator state.

Arguments

<i>pstate</i>	Pointer to the <code>ccGenState</code> object to clean up.

Return Value

None

Miscellaneous Functions

ccFreePathWidths

```
void  
ccFreePathWidths(  
    const ccPathWidthSpec *pathSpec  
);
```

Description

(ICADVM20.1 Photonics Only) Frees the memory allocated for the given ccPathWidthSpec data. This function is often used by [ccGetPathWidths](#).

Note: This function does not control or deallocate the memory allocated for the specification .

Arguments

<i>pathSpec</i>	Pointer to a ccPathWidthSpec.

Return Value

None

ccReleaseObject

```
void  
ccReleaseObject(  
    ccId id  
);
```

Description

(ICADV20.1 Photonics Only) Releases the object specified by the ccID. Additionally:

- For mathematical objects, such as ccCurve, ccFacet, ccPolyCurve, ccBoundary, it frees the underlying object. Therefore, for every call of a function that provides ID, such as [ccGetPolyCurve](#) and ccGen* there should be a matching ccReleaseObject call.
- For ccShapes, ccReleaseObject decreases the reference count by 1, so for every [ccGetFromFig](#) call there should be a matching ccReleaseObject call.

Arguments

<i>id</i>	ID of a CurvyCore object.

Return Value

None

Waveguide Functions

This section covers the following topics and waveguide functions.

- [Structs in the Waveguide Infrastructure](#)
- [Enumerated Types and Enumerated Constants used in the Waveguide Infrastructure](#)
- [techAllocWaveguideModePropArray](#)
- [techAllocWGDerivedShapeSpecArray](#)
- [techCreateWaveguideDef](#)
- [techDeleteWaveguideDef](#)
- [techFindWaveguideDef](#)
- [techFreeWaveguideModePropArray](#)
- [techFreeWGDerivedShapeSpecArray](#)
- [techGenWaveguideDef](#)
- [techGetWaveguideDefDerivedShapeSpecs](#)
- [techGetWaveguideDefLayer](#)
- [techGetWaveguideDefMinBendRadius](#)
- [techGetWaveguideDefMinWidth](#)
- [techGetWaveguideDefName](#)
- [techGetWaveguideDefPurpose](#)
- [techStartGenWaveguideDef](#)
- [techHasWaveguideDefMinBendRadius](#)
- [techSetWaveguideDefMinBendRadius](#)
- [techUnsetWaveguideDefMinBendRadius](#)

Note: You need the `Virtuoso_Photonics_Option` license to use the functionality discussed in this section. For information about obtaining the required license, contact your local Cadence representative.

Structs in the Waveguide Infrastructure

This section covers the structs used in the Waveguide infrastructure.

techWGDerivedShapeSpec

Type	Name	Description
<i>techLayerNum</i>	<i>layerNum</i>	Layer number associated with WGDerivedShape
<i>techPurpose</i>	<i>purposeNum</i>	Purpose associated with WGDerivedShape
<i>techWGDerivedShapeType</i>	<i>shapeType</i>	Type of shape
<i>techWGEnclosureValueType</i>	<i>encValueType</i>	Integer or string specifying the type of enclosure of WGDerivedShape.
<i>techWGEnclosureValueType</i>	<i>innerEncValueType</i>	Integer or string specifying the type of inner enclosure of WGDerivedShape
<i>techWGEnclosureValueType</i>	<i>outerEncValueType</i>	Integer or string specifying the type of outer enclosure of WGDerivedShape

Enumerated Types and Enumerated Constants used in the Waveguide Infrastructure

Waveguide side direction

Direction	Enumerated constants
Both	0
Right	1
Left	2

Waveguide derived shape type

Shape Type	Enumerated constants
Enclosure	0
Offset	1

Waveguide enclosure value type

Enclosure value type	Enumerated constants
Unspecified	0
Integer	1
String	2

techAllocWaveguideModePropArray

```
techWGModeProperty*  
techAllocWaveguideModePropArray(  
    unsigned int numProps  
);
```

Description

Allocates memory for the techWGModeProperty array. All members of the array are initialized to zero or NULL.

Arguments

<i>numProps</i>	Number of techWGModeProperty specifications that can fit in the allocated memory.

Return Value

<i>techWGModeProperty</i>	Pointer to the allocated memory area.

techAllocWGDerivedShapeSpecArray

```
techWGDerivedShapeSpec*  
techAllocWGDerivedShapeSpecArray(  
    unsigned int numSpecs  
);
```

Description

Allocates memory for the techWGDerivedShapeSpec array. All members of the array are initialized to zero or NULL.

Arguments

<i>numSpecs</i>	Number of WGDerivedShape specifications that can fit in the allocated memory.

Return Value

<i>techWGDerivedShapeSpec</i>	Pointer to the allocated memory area.

techCreateWaveguideDef

```
techWaveguideDefId  
techCreateWaveguideDef(  
    techFileId                techfile,  
    constString               name,  
    techLayerNum              layerNum,  
    techPurpose                purposeNum,  
    dbDistance                minWidth,  
    dbDistance                minBendRadius,  
    unsigned int              numShapeSpecs,  
    const techWGDerivedShapeSpec *shapeSpecs,  
    const techWGModePropertySpec *modeProperties  
  
    );
```

Description

Creates a waveguideDef in the specified technology file and returns its ID.

Arguments

<i>techfile</i>	Database ID of the technology file in which the waveguideDef object is to be created.
<i>name</i>	Unique name of the new waveguideDef object.
<i>layerNum</i>	Layer number of the waveguideDef object.
<i>purposeNum</i>	Purpose number associated with the waveguideDef object.
<i>minWidth</i>	Minimum width of the object.
<i>minBendRadius</i>	Minimum bend radius of the object.
<i>numShapeSpecs</i>	Number of shape specifications to create the waveguideDef object.
<i>*shapeSpecs</i>	Pointer to the shapeSpecs array.
<i>*modeProperties</i>	Pointer to the array of mode property specifications.

Return Value

techWaveguideDefId

Cadence Integrators Toolkit Database Reference

CDBA Photonics Functions

ID of the new `waveguideDef` object created in the specified technology file.

techDeleteWaveguideDef

Boolean

```
techDeleteWaveguideDef(  
    techWaveguideDefId    wgDefId  
);
```

Description

Deletes the specified waveguideDef.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.

Return Value

<i>Boolean</i>	TRUE if the specified waveguideDef object is deleted. FALSE if the operation fails.

techFindWaveguideDef

```
techWaveguideDefId  
techFindWaveguideDef(  
    techFileId      techfile,  
    techLayerNum    layerNum,  
    techPurpose     purposeNum  
);
```

Description

Searches for the specified waveguideDef object in the given technology database and returns its ID.

Arguments

<i>techfile</i>	Database ID of the technology file.
<i>layerNum</i>	Layer number for the waveguideDef object.
<i>purposeNum</i>	Purpose number associated with the waveguideDef object.

Return Value

<i>techWaveguideDefId</i>	ID of the waveguideDef object, if found.
	Returns <code>NULL</code> if the specified waveguideDef object is not found.

techFreeWaveguideModePropArray

```
void  
techFreeWaveguideModePropArray(  
    unsigned int numProps,  
    techWGDModeProperty *props  
);
```

Description

Frees the memory allocated for a techWGDModeProperty array.

Arguments

<i>numProps</i>	Size of the deallocated memory area.
<i>*props</i>	Pointer to the memory area to deallocate.

Return Value

None

techFreeWGDerivedShapeSpecArray

```
void  
techFreeWGDerivedShapeSpecArray(  
    unsigned int numSpecs,  
    techWGDerivedShapeSpec  *specs  
);
```

Description

Frees the memory allocated for a techWGDerivedShapeSpec array.

Arguments

<i>numSpecs</i>	Size of the deallocated memory area.
<i>*specs</i>	Pointer to the memory area to deallocate.

Return Value

None

techGenWaveguideDef

Boolean

```
techGenWaveguideDef(  
    techGenStateId    pState,  
    techWaveguideDefId *wgDefId  
);
```

Description

Retrieves the ID of the next waveguideDef object from the generator with the specified *pState*. ID of the waveguideDef is returned in *wgDefId*.

Arguments

<i>pState</i>	State ID for the generator.
<i>*wgDefId</i>	ID of the next waveguideDef object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	TRUE if the next waveguideDef object ID is successfully generated. FALSE if the list is exhausted or an error occurs.

techGetWaveguideDefDerivedShapeSpecs

```
void  
techGetWaveguideDefDerivedShapeSpecs (  
    techWaveguideDefId      wgDefId,  
    unsigned int            *numShapeSpecs,  
    techWGDerivedShapeSpec **shapeSpecs  
);
```

Description

Retrieves the derivedShapeSpecs for the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.
<i>*numShapeSpecs</i>	Number of entries in the array of derivedShapeSpecs.
<i>**shapeSpecs</i>	Pointer to the memory area where the array of derivedShapeSpecs is placed.

Return Value

None

techGetWaveguideDefLayer

```
techLayerNum  
techGetWaveguideDefLayer(  
    techWaveguideDefId    wgDefId  
);
```

Description

Returns the layer number of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.

Return Value

<i>techLayerNum</i>	Layer number of the specified waveguideDef object.
	Returns <code>techcNullLayer</code> in case of an error.

techGetWaveguideDefMinBendRadius

```
dbDistance  
techGetWaveguideDefMinBendRadius (  
    techWaveguideDefId wgDefId  
);
```

Description

Returns the minimum bend radius of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.

Return Value

<i>dbDistance</i>	Minimum bend radius of the specified waveguideDef object. Returns 0 in case of an error.

techGetWaveguideDefMinWidth

```
dbDistance  
techGetWaveguideDefMinWidth(  
    techWaveguideDefId wgDefId  
);
```

Description

Returns the minimum width of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of the specified waveguideDef object.

Return Value

<i>dbDistance</i>	Minimum width of the specified waveguideDef object. Returns 0 in case of an error.

techGetWaveguideDefName

String

```
techGetWaveguideDefName (  
    techWaveguideDefId    wgDefId  
);
```

Description

Returns the name of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of the specified waveguideObject.

Return Value

<i>String</i>	Name of the specified waveguideDef object. Returns <code>NULL</code> in case of an error.

techGetWaveguideDefPurpose

```
techPurpose  
techGetWaveguideDefPurpose(  
    techWaveguideDefId wgDefId  
);
```

Description

Returns the purpose number of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of the specified waveguideObject.

Return Value

<i>techPurpose</i>	Purpose number of the specified waveguideObject. Returns <code>techcUnknownPurpose</code> in case of an error.

techStartGenWaveguideDef

```
techGenStateId  
techStartGenWaveguideDef(  
    techFileId  techfile  
);
```

Description

Initializes a generator for all the waveguideDef objects in the specified technology file. A generator state ID is returned, which is used with the [techGenWaveguideDef](#) function.

Arguments

<i>techfile</i>	Database ID of the technology file.

Return Value

<i>techGenStateId</i>	State ID for the generator.

techHasWaveguideDefMinBendRadius

Boolean

```
techHasWaveguideDefMinBendRadius (  
    techWaveguideDefId wgDefId,  
);
```

Description

Checks if the specified waveguideDef object has the minimum bend radius.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.

Return Value

<i>Boolean</i>	TRUE if the waveguideDef object has the minimum bend radius. FALSE if the waveguideDef object does not have the minimum bend radius.

techSetWaveguideDefMinBendRadius

Boolean

```
techSetWaveguideDefMinBendRadius (  
    techWaveguideDefId wgDefId,  
    dbDistance minBendRadius  
);
```

Description

Sets the minimum bend radius of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.
<i>minBendRadius</i>	Minimum bend radius of the object.

Return Value

<i>Boolean</i>	TRUE if the minimum bend radius is set. FALSE if the minimum bend radius is not set.

techUnsetWaveguideDefMinBendRadius

Boolean

```
techUnsetWaveguideDefMinBendRadius (  
    techWaveguideDefId wgDefId  
);
```

Description

Unsets the minimum bend radius of the specified waveguideDef object.

Arguments

<i>wgDefId</i>	ID of a waveguideDef object.

Return Value

<i>Boolean</i>	TRUE if the minimum bend radius is reset. FALSE if the minimum bend radius is not reset.

Snap Pattern Definition Functions

A SnapPattern object is a technology database object that provides an improved control over the layout of instances for some layer-purpose pairs (LPPs) in advanced node designs. The snap pattern data is stored in a Snap Pattern Definition (SnapPatternDef) object.

You can use the SnapPatternDef functions to create, search, and delete SnapPatternDefs and to modify the underlying SnapPatterns data.

To learn how to use SnapPatternDef functions, see [Example- Using Snap Pattern Definition Functions](#).

Supported functions include:

- [techAllocEncSnappingLayerArray](#)
- [techAllocSnappingLayerArray](#)
- [techFreeEncSnappingLayerArray](#)
- [techFreeSnappingLayerArray](#)
- [techCreateSnapPatternDef](#)
- [techCreateSnapPatternDefWithTrackGroups](#)
- [techCreateSnapPatternDef](#)
- [techDeleteSnapPatternDef](#)
- [techFindSnapPatternDef](#)
- [techFindSnapPatternDefByLPP](#)
- [techFindSnapPatternDefByName](#)
- [techGenSnapPatternDef](#)
- [techGenSnapPatternDefForLPP](#)
- [techGetSnapPatternDefLayer](#)

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

- [techGetSnapPatternDefName](#)
- [techGetSnapPatternDefOffset](#)
- [techGetSnapPatternDefPurpose](#)
- [techGetSnapPatternDefSnappingLayers](#)
- [techGetSnapPatternDefStep](#)
- [techGetSnapPatternDefTrackGroups](#)
- [techGetSnapPatternDefTrackWidth](#)
- [techGetSnapPatternDefType](#)
- [techIsSnapPatternDefHorizontal](#)
- [techSnapPatternDefTypeToString](#)
- [techStartGenSnapPatternDef](#)
- [techStartGenSnapPatternDefForLPP](#)
- [techStringToSnapPatternDefType](#)

techAllocEncSnappingLayerArray

```
techEncSnappingLayer*  
techAllocEncSnappingLayerArray(  
    unsigned int numLayers  
);
```

Description

Allocates memory for the array of enclosed snapping layers and initializes the memory with zeroes.

Arguments

<i>numLayers</i>	Number of enclosed snapping layer specifications to fit in the allocated memory.

Return Value

<i>techEncSnappingLayer*</i>	Pointer to the allocated memory area.

techAllocSnappingLayerArray

```
techSnappingLayer*  
techAllocSnappingLayerArray(  
    unsigned int    numLayers  
);
```

Description

Allocates memory for the array of snapping layers and initializes the memory with zeros.

Arguments

<i>numLayers</i>	Number of snapping layer specifications to fit in the allocated memory.

Return Value

<i>techSnappingLayer*</i>	Pointer to the allocated memory area. Returns <code>NULL</code> in case of failure.

techFreeEncSnappingLayerArray

```
void  
techFreeEncSnappingLayerArray(  
    unsigned int      numLayers,  
    techEncSnappingLayer *layers  
);
```

Description

Deallocates the memory allocated for an array of enclosed snapping layers.

Note: In addition to deallocating the specified memory area, the function also frees the memory internally on the non-null `enclosures` and `snappingLayer.purposes` members of each enclosed snapping layer in the array.

Arguments

<i>numLayers</i>	Size of the deallocated memory area, measured in blocks of <code>sizeof(techEncSnappingLayers)</code> bytes.
<i>layers</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

techCreateSnapPatternDef

techSnapPatternDefId

```
techSnapPatternDefId
techCreateSnapPatternDef(
    techFileId techId,
    String name,
    techLayerNum layerNum,
    techPurpose purposeNum,
    Boolean isHorizontal,
    dbDistance step,
    techEncSnappingLayer *snappingLayers,
    unsigned int numSnappingLayers,
    techSnapPatternDefType type,
    dbDistance offset,
    dbDistance trackWidth
);
```

Description

Creates a snap pattern definition object in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the snap pattern definition object is created.
<i>name</i>	Name for the snap pattern definition object.
<i>layerNum</i>	Layer number for the snap pattern definition object.
<i>purposeNum</i>	Purpose number for the snap pattern definition object.
<i>isHorizontal</i>	Pattern direction: TRUE for horizontal and FALSE for vertical.
<i>step</i>	Distance between the pattern tracks.
<i>snappingLayers</i>	Array of snapping layers for the snap pattern definition object.
<i>numSnappingLayers</i>	Number of elements in the array of snapping layers.
<i>type</i>	Type of the snap pattern definition object.
<i>offset</i>	Distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

trackWidth Width of the physical shape, represented by each track.

Return Value

techSnapPatternDefId ID of the created snap pattern definition object.

techCreateSnapPatternDefWithTrackGroups

```
techSnapPatternDefId
    techCreateSnapPatternDefWithTrackGroups (
        techFileId techId,
        String name,
        techLayerNum layerNum,
        techPurpose purposeNum,
        Boolean isHorizontal,
        dbDistance step,
        techEncSnappingLayer *snappingLayers,
        unsigned int numSnappingLayers,
        techSnapPatternDefType type,
        dbDistance offset,
        dbDistance trackWidth,
        const techSnapPatternDefTrackGroup *trackGroups,
        unsigned int numTrackGroups
    );
```

Description

Creates a snap pattern definition object with track groups in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the snap pattern definition object is created.
<i>name</i>	Name for the snap pattern definition object.
<i>layerNum</i>	Layer number for the snap pattern definition object.
<i>purposeNum</i>	Purpose number for the snap pattern definition object.
<i>isHorizontal</i>	Pattern direction: TRUE for horizontal and FALSE for vertical.
<i>step</i>	Distance between the pattern tracks.
<i>snappingLayers</i>	Array of snapping layers for the snap pattern definition object.
<i>numSnappingLayers</i>	Number of elements in the array of snapping layers.
<i>type</i>	Type of the snap pattern definition object.
<i>offset</i>	Distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

<i>trackWidth</i>	Width of the physical shape, represented by each track.
<i>trackGroups</i>	Array of track groups.
<i>numTrackGroups</i>	Number of elements in the array of track groups. A warning is displayed if the function is called with <i>numTrackGroups</i> = 0.

Return Value

<code>techSnapPatternDefId</code>	ID of the created snap pattern definition object.

techFreeSnappingLayerArray

```
void  
techFreeSnappingLayerArray(  
    unsigned int      numLayers,  
    techSnappingLayer *layers  
);
```

Description

Deallocates the memory allocated for an array of snapping layers.

Note: In addition to deallocating the specified memory area, the function also frees the memory internally on the non-null `purposes` members of each snapping layer in the array.

Arguments

<i>numLayers</i>	Size of the deallocated memory area, measured in blocks of <code>sizeof(dbSnappingLayer)</code> bytes.
<i>layers</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

techCreateSnapPatternDef

```
techSnapPatternDefId
techSnapPatternDefId
    techCreateSnapPatternDef(
        techFileId techId,
        String name,
        techLayerNum layerNum,
        techPurpose purposeNum,
        Boolean isHorizontal,
        dbDistance step,
        techEncSnappingLayer *snappingLayers,
        unsigned int numSnappingLayers,
        techSnapPatternDefType type,
        dbDistance offset,
        dbDistance trackWidth
    );
```

Description

Creates a snap pattern definition object in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the snap pattern definition object is created.
<i>name</i>	Name for the snap pattern definition object.
<i>layerNum</i>	Layer number for the snap pattern definition object.
<i>purposeNum</i>	Purpose number for the snap pattern definition object.
<i>isHorizontal</i>	Pattern direction: TRUE for horizontal and FALSE for vertical.
<i>step</i>	Distance between the pattern tracks.
<i>snappingLayers</i>	Array of snapping layers for the snap pattern definition object.
<i>numSnappingLayers</i>	Number of elements in the array of snapping layers.
<i>type</i>	Type of the snap pattern definition object.
<i>offset</i>	Distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

trackWidth Width of the physical shape, represented by each track.

Return Value

techSnapPatternDefId ID of the created snap pattern definition object.

techCreateSnapPatternDefWithTrackGroups

```
techSnapPatternDefId  
techCreateSnapPatternDefWithTrackGroups (  
    techFileId techId,  
    String name,  
    techLayerNum layerNum,  
    techPurpose purposeNum,  
    Boolean isHorizontal,  
    dbDistance step,  
    techEncSnappingLayer *snappingLayers,  
    unsigned int numSnappingLayers,  
    techSnapPatternDefType type,  
    dbDistance offset,  
    dbDistance trackWidth,  
    const techSnapPatternDefTrackGroup *trackGroups,  
    unsigned int numTrackGroups  
);
```

Description

Creates a snap pattern definition object with track groups in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the snap pattern definition object is created.
<i>name</i>	Name for the snap pattern definition object.
<i>layerNum</i>	Layer number for the snap pattern definition object.
<i>purposeNum</i>	Purpose number for the snap pattern definition object.
<i>isHorizontal</i>	Pattern direction: TRUE for horizontal and FALSE for vertical.
<i>step</i>	Distance between the pattern tracks.
<i>snappingLayers</i>	Array of snapping layers for the snap pattern definition object.
<i>numSnappingLayers</i>	Number of elements in the array of snapping layers.
<i>type</i>	Type of the snap pattern definition object.
<i>offset</i>	Distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

<i>trackWidth</i>	Width of the physical shape, represented by each track.
<i>trackGroups</i>	Array of track groups.
<i>numTrackGroups</i>	Number of elements in the array of track groups. A warning is displayed if the function is called with <i>numTrackGroups</i> = 0.

Return Value

<code>techSnapPatternDefId</code>	ID of the created snap pattern definition object.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

techDeleteSnapPatternDef

Boolean

```
techDeleteSnapPatternDef(  
    techSnapPatternDefId  SPDef  
);
```

Description

Deletes the specified snap pattern definition object.

Arguments

<i>SPDef</i>	ID of the snap pattern definition object to be deleted.

Return Values

<i>Boolean</i>	TRUE, if the object was successfully deleted.
	FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

techFindSnapPatternDef

```
techSnapPatternDefId  
techFindSnapPatternDef(  
    techFileId      techId,  
    techLayerNum    layerNum,  
    techPurpose     purposeNum  
);
```

Description

Searches for the snap pattern definition object by the specified layer and purpose in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the search is performed.
<i>layerNum</i>	Layer number in a layer-purpose pair.
<i>purposeNum</i>	Purpose number in a layer-purpose pair.

Return Value

<i>techSnapPatternDefId</i>	ID of the snap pattern definition object, if found. Returns <code>NULL</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

techFindSnapPatternDefByLPP

```
techSnapPatternDefId  
techFindSnapPatternDefByLPP(  
    techFileId    techId,  
    techLPId      lpId  
);
```

Description

Searches for the snap pattern definition object by the specified layer-purpose pair (LPP) in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the search is performed.
<i>lpId</i>	ID of the LPP whose layer and purpose IDs are used for the search.

Return Value

<i>techSnapPatternDefId</i>	ID of the snap pattern definition object, if found. Returns <code>NULL</code> in case of failure.

techFindSnapPatternDefByName

```
techSnapPatternDefId  
techFindSnapPatternDefByName (  
    techFileId          techId,  
    cppConstString      name  
);
```

Description

Searches for the snap pattern definition object with the specified name in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the search is performed.
<i>name</i>	Name of the snap pattern definition object for which the search is performed.

Return Value

<i>techSnapPatternDefId</i>	ID of the snap pattern definition object, if found. Returns <code>NULL</code> in case of failure.

techFindSnapPatternDefByName

```
techSnapPatternDefId  
techFindSnapPatternDefByName (  
    techFileId          techId,  
    cppConstString      name  
);
```

Description

Searches for the snap pattern definition object with the specified name in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the search is performed.
<i>name</i>	Name of the snap pattern definition object for which the search is performed.

Return Value

<i>techSnapPatternDefId</i>	ID of the snap pattern definition object, if found. Returns <code>NULL</code> in case of failure.

techGenSnapPatternDef

Boolean

```
techGenSnapPatternDef(  
    techGenStateId      pState,  
    techSnapPatternDefId *snapPatternDefId  
);
```

Description

Retrieves the ID of the next snap pattern definition object from the generator with the given `techGenStateId`. The snap pattern definition ID is returned in the `snapPatternDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>snapPatternDefId</i>	ID of the next snap pattern definition produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the next snap pattern definition ID is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

techGenSnapPatternDefForLPP

Boolean

```
techGenSnapPatternDefForLPP(  
    techGenStateId      pState,  
    techSnapPatternDefId *snapPatternDefId  
);
```

Description

Retrieves the ID of the next snap pattern definition object from the generator with the given state. The snap pattern definition ID is returned in the `snapPatternDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>snapPatternDefId</i>	ID of the next snap pattern definition object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the next snap pattern definition ID is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

techGetSnapPatternDefLayer

```
techLayerNum  
techGetSnapPatternDefLayer(  
    techSnapPatternDefId    SPDef  
);
```

Description

Retrieves the layer number for the specified snap pattern definition object.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.

Return Value

<i>techLayerNum</i>	Layer number for the specified snap pattern definition object. Returns <code>techcNullLayer</code> in case of error.

techGetSnapPatternDefName

String

```
techGetSnapPatternDefName(  
    techSnapPatternDefId    SPDef  
);
```

Description

Retrieves the name for the specified snap pattern definition object.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.

Return Value

<i>String</i>	Name of the specified snap pattern definition object. Returns <code>NULL</code> in case of error.

techGetSnapPatternDefOffset

```
dbDistance  
techGetSnapPatternDefOffset(  
    techSnapPatternDefId  SPDef  
);
```

Description

Retrieves the offset for the specified snap pattern definition object. Here, offset is the distance of the first track from the period track.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.

Return Value

<i>dbDistance</i>	Offset value of the specified snap pattern definition object. Returns 0 in case of error.

techGetSnapPatternDefPurpose

```
techPurpose  
techGetSnapPatternDefPurpose (  
    techSnapPatternDefId SPDef  
);
```

Description

Retrieves the purpose for the specified snap pattern definition object.

Arguments

<i>spDef</i>	ID of a snap pattern definition object.

Return Value

<i>techPurpose</i>	Purpose of the specified snap pattern definition object. Returns <code>techcUnknownPurpose</code> in case of error.

techGetSnapPatternDefSnappingLayers

```
void  
techGetSnapPatternDefSnappingLayers (  
    techSnapPatternDefId      SPDef,  
    unsigned int              *numSnappingLayers,  
    techEncSnappingLayer      **snappingLayers  
);
```

Description

Allocates a suitable amount of heap memory, and fills it with an array of snapping layers of the specified snap pattern definition object. Pointer to the allocated memory area is returned through the *snappingLayers* parameter and the size of this memory area is returned through the *numSnappingLayers* parameter.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.
<i>numSnappingLayers</i>	Pointer to the memory area in which the number of snapping layers is placed.
<i>snappingLayers</i>	Pointer to the memory area in which the array of snapping layers is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

techGetSnapPatternDefStep

```
dbDistance  
techGetSnapPatternDefStep(  
    techSnapPatternDefId    SPDef  
);
```

Description

Retrieves the step (spacing between pattern tracks) for the specified snap pattern definition.

Arguments

<i>wspId</i>	ID of a snap pattern definition object.

Return Value

<i>dbDistance</i>	Step value of the specified snap pattern definition object. Returns 0 in case of error.

techGetSnapPatternDefTrackGroups

void

```
techGetSnapPatternDefTrackGroups (  
    techSnapPatternDefId SPdef,  
    unsigned int *numTrackGroups,  
    techSnapPatternDefTrackGroup **trackGroups  
);
```

Description

Retrieves the list of track groups from the specified snap pattern definition object.

Arguments

<i>SPdef</i>	ID of a snap pattern definition object.
<i>numTrackGroups</i>	Pointer to the memory area in which the number of track groups is placed.
<i>trackGroups</i>	Pointer to the memory area in which the array of track group entries is placed.

Return Value

No value returned.

techGetSnapPatternDefTrackWidth

dbDistance

```
techGetSnapPatternDefTrackWidth(  
    techSnapPatternDefId  SPDef  
);
```

Description

Retrieves the track width for the specified snap pattern definition object.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.

Return Value

<i>dbDistance</i>	Track width of the specified snap pattern definition object. Returns 0 in case of error.

techGetSnapPatternDefType

```
techSnapPatternDefType  
techGetSnapPatternDefType (  
    techSnapPatternDefId  SPDef  
);
```

Description

Retrieves the type for the specified snap pattern definition object. Valid values are 0 and 1, or integer constants `techcSPTypeLocal` and `techcSPTypeGlobal`, respectively.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.

Return Value

techSnapPatternDefType Type of the specified snap pattern definition object.

techIsSnapPatternDefHorizontal

Boolean

```
techIsSnapPatternDefHorizontal(  
    techSnapPatternDefId    SPDef  
);
```

Description

Verifies whether the specified snap pattern definition object direction is horizontal.

Arguments

<i>SPDef</i>	ID of a snap pattern definition object.

Return Value

<i>Boolean</i>	TRUE, if the object is horizontal.
	FALSE, if the object is vertical.

techSnapPatternDefTypeToString

String

```
techSnapPatternDefTypeToString(  
    int  type  
);
```

Description

Converts a `techSnapPatternDefType` enum to a string holding the name of the corresponding `snapPatternDef` type. Valid input values are 0 and 1, or integer constants `techcSPTypeLocal` and `techcSPTypeGlobal`, respectively.

Arguments

<i>type</i>	A snap pattern definition type enum.

Return Value

<i>String</i>	"local", if input value is 0 or <code>techcSPTypeLocal</code> . "global", if input value is 1 or <code>techcSPTypeGlobal</code> . Returns NULL in case of failure.

techStartGenSnapPatternDef

```
techGenStateId  
  
techStartGenSnapPatternDef(  
    techFileId  techId  
);
```

Description

Initializes a generator for all snap pattern definition objects in the specified technology file. A generator state is returned, which is used with the [techGenSnapPatternDef](#) function.

Arguments

<i>techId</i>	ID of the technology file containing the snap pattern definition objects to be generated.

Return Value

<i>techGenStateId</i>	State ID for the generator.

techStartGenSnapPatternDefForLPP

```
techGenStateId  
  
techStartGenSnapPatternDefForLPP(  
    techFileId  techId,  
    techLPId    snapLPId  
);
```

Description

Initializes a generator for all snap pattern definition objects in the specified technology file that have the specified snapping LPP. A generator state is returned, which is used with the [techGenSnapPatternDefForLPP](#) function.

Arguments

<i>techId</i>	ID of the technology file containing the snap pattern definition objects to be generated.
<i>snapLPId</i>	ID of the LPP used for generation.

Return Value

<i>techGenStateId</i>	State ID for the generator.

techStringToSnapPatternDefType

Boolean

```
techStringToSnapPatternDefType(  
    cppConstString    typeStr,  
    techSnapPatternDefType *type  
);
```

Description

Converts the string holding a snap pattern definition type name to a `techSnapPatternDefType` enum. Valid input strings are "local" and "global". The result is written to the memory area referenced by the `type` pointer.

Arguments

<i>typeStr</i>	A snap pattern definition type name.
<i>type</i>	Pointer to the memory area where the conversion result will be written. The value referenced by the pointer is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	TRUE, if the conversion returned a valid result. FALSE, in case of failure.

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

Example- Using Snap Pattern Definition Functions

```
// Create the test data.
ddId          lib = ddCreateLib("design", getTestDir() + "/design");
techFileId    techfile = techOpenTechFile("design", "tech.db", "a");

tcLoadTechFile(techfile, "dump.tf", "w", ilcNil);

constString    name = "finfet1";
techLayerNum   layerNum = 4;
techLayerNum   snapLayerNum = 5;
techPurpose    purposeNum = 1001;
Boolean        isHorizontal = TRUE;
dbDistance    offset = 30;
dbDistance    step = 20;
unsigned int   numTrackEnclosures = 2;
unsigned int   numSnapPurposes = 1;
unsigned int   snapPurposeNum = 5;
dbDistance    trackWidth = 19;
techEncSnappingLayer *snappingLayers = techAllocEncSnappingLayerArray(1);
techSnapPatternDefType type = techcSPTTypeGlobal;

snappingLayers[0].enclosures = calloc(numTrackEnclosures,
                                       dbDualDistance);
snappingLayers[0].snappingLayer.purposes = calloc(numSnapPurposes,
                                                    techPurpose);

snappingLayers[0].snappingLayer.layerNum = snapLayerNum;
snappingLayers[0].enclosures[0].first = 1;
snappingLayers[0].enclosures[0].second = 2;
snappingLayers[0].enclosures[1].first = 3;
snappingLayers[0].enclosures[1].second = 4;
snappingLayers[0].numEnclosures = numTrackEnclosures;
snappingLayers[0].snappingLayer.purposes[0] = snapPurposeNum;
snappingLayers[0].snappingLayer.numPurposes = numSnapPurposes;

techSnapPatternDefId spId = techCreateSnapPatternDef(techfile,
                                                    (String) name,
                                                    layerNum,
                                                    purposeNum,
                                                    isHorizontal,
```

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

```
        step,
        snappingLayers,
        1,
        type,
        offset,
        trackWidth);

// Examples for the getter functions.
cout << "Sample snapPatternDef dump: " << endl;
cout << "  name: "
    << techGetSnapPatternDefName(spId) << endl;
cout << "  layer: "
    << techGetSnapPatternDefLayer(spId) << endl;
cout << "  purpose: "
    << techGetSnapPatternDefPurpose(spId) << endl;
cout << "  horizontal: "
    << (bool) techIsSnapPatternDefHorizontal(spId) << endl;
cout << "  offset: "
    << techGetSnapPatternDefOffset(spId) << endl;
cout << "  step: "
    << techGetSnapPatternDefStep(spId) << endl;
cout << "  track width: "
    << techGetSnapPatternDefTrackWidth(spId) << endl;
cout << "  type: "
    << techSnapPatternDefTypeToString(techGetSnapPatternDefType(spId))
    << endl;

unsigned int          tmp_num = 0;
techEncSnappingLayer *tmp_snapLayers = NULL;

techGetSnapPatternDefSnappingLayers(spId, &tmp_num, &tmp_snapLayers);

cout << "  snapping layers:" << endl;

assert(tmp_num == 1);
assert(tmp_snapLayers[0].snappingLayer.layerNum
    == snappingLayers[0].snappingLayer.layerNum);
assert(tmp_snapLayers[0].numEnclosures == snappingLayers[0].numEnclosures);
assert(tmp_snapLayers[0].snappingLayer.numPurposes
    == snappingLayers[0].snappingLayer.numPurposes);

cout << "    layer: " << tmp_snapLayers[0].snappingLayer.layerNum << " purposes: (";
```

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

```
for (oaUInt4 k = 0; k < snappingLayers[0].snappingLayer.numPurposes; k++) {
    assert(tmp_snapLayers[0].snappingLayer.purposes[k]
           == snappingLayers[0].snappingLayer.purposes[k]);

    cout << " " << tmp_snapLayers[0].snappingLayer.purposes[k];
}

cout << " )" << endl << endl;

techFreeEncSnappingLayerArray(1, tmp_snapLayers);

// Examples for the generator functions.
techGenStateId      gen_state = techStartGenSnapPatternDef(techfile);
techSnapPatternDefId tmp_spDef;

cout << "Snap pattern definition names returned by the techGenSnapPatternDef
generator: ";

while (techGenSnapPatternDef(gen_state, &tmp_spDef))
    cout << " " << techGetSnapPatternDefName(tmp_spDef);

cout << endl << endl;
techStopGen(gen_state);

cout << "Snap pattern definition names returned by the techGenSnapPatternDefForLPP
generator: ";

techLPId      lpp = techCreateLP(techfile, layerNum,
                                purposeNum, "M3");
techLPId      snap_lpp = techCreateLP(techfile, snapLayerNum,
                                      snapPurposeNum, "VIA3");

assert(lpp);

gen_state = techStartGenSnapPatternDefForLPP(techfile, snap_lpp);

while (techGenSnapPatternDefForLPP(gen_state, &tmp_spDef))
    cout << " " << techGetSnapPatternDefName(tmp_spDef);

cout << endl << endl;
```

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

```
techStopGen(gen_state);
```

```
// Examples for the find and delete functions.
```

```
assert(spId == techFindSnapPatternDefByLPP(techfile, lpp));
```

```
assert(spId == techFindSnapPatternDef(techfile, layerNum, purposeNum));
```

```
assert(spId == techFindSnapPatternDefByName(techfile, name));
```

```
assert(techDeleteSnapPatternDef(spId));
```

```
assert(techFindSnapPatternDefByName(techfile, name) ==  
techcNullSnapPatternDefId);
```

```
// Clean up the test data.
```

Cadence Integrators Toolkit Database Reference

Snap Pattern Definition Functions

Width Spacing Pattern Functions

This chapter includes the following topics:

- WSP Technology File Functions
- WSP Database Functions

WSP Technology File Functions

Width Spacing Pattern (WSP) technology file functions are used to operate with `widthSpacingPattern`, `widthSpacingPatternGroup`, `widthSpacingSnapPatternDef`, and `relatedSnapPatterns` objects in the technology database.

These functions have the ability to create, find, and delete objects, access object parameters, and iterate over the objects stored in the database.

To learn how to use WSP technology file functions, see [Example- Using the WSP Technology File Functions](#).

Supported functions include:

- [Miscellaneous Technology File Functions](#)
 - ❑ [techMaskColorTypeToString](#)
 - ❑ [techStringToMaskColorType](#)
- [Width Spacing Pattern Functions](#)
 - ❑ [techAllocWidthSpacingPatternSpecArray](#)
 - ❑ [techCreateWidthSpacingPattern](#)
 - ❑ [techDeleteWidthSpacingPattern](#)
 - ❑ [techFindWidthSpacingPattern](#)
 - ❑ [techFreeWidthSpacingPatternSpecArray](#)
 - ❑ [techGenWidthSpacingPattern](#)
 - ❑ [techGetWidthSpacingPatternAllowedRepeatMode](#)
 - ❑ [techGetWidthSpacingPatternDefaultRepeatMode](#)
 - ❑ [techGetWidthSpacingPatternName](#)
 - ❑ [techGetWidthSpacingPatternOffset](#)
 - ❑ [techGetWidthSpacingPatternPatterns](#)
 - ❑ [techGetWidthSpacingPatternRepeatOffset](#)
 - ❑ [techGetWidthSpacingPatternShiftColor](#)
 - ❑ [techGetWidthSpacingPatternStartingColor](#)

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

- ☐ [techSetWidthSpacingPatternRepeatMode](#)
- ☐ [techGetWidthSpacingPatternStartingColor](#)
- ☐ [techStartGenWidthSpacingPattern](#)

■ [Width Spacing Pattern Group Functions](#)

- ☐ [techCreateWidthSpacingPatternGroup](#)
- ☐ [techDeleteWidthSpacingPatternGroup](#)
- ☐ [techFindWidthSpacingPatternGroup](#)
- ☐ [techGenWidthSpacingPatternGroup](#)
- ☐ [techGetWidthSpacingPatternGroupName](#)
- ☐ [techGetWidthSpacingPatternGroupPatternNames](#)
- ☐ [techStartGenWidthSpacingPatternGroup](#)

■ [Width Spacing Snap Pattern Definition Functions](#)

- ☐ [techCreateWidthSpacingSnapPatternDef](#)
- ☐ [techDeleteWidthSpacingSnapPatternDef](#)
- ☐ [techFindWidthSpacingSnapPatternDefByName](#)
- ☐ [techGenWidthSpacingSnapPatternDef](#)
- ☐ [techGenWidthSpacingSnapPatternDefForLPP](#)
- ☐ [techGetWidthSpacingSnapPatternDefDefaultActiveName](#)
- ☐ [techGetWidthSpacingSnapPatternDefLayer](#)
- ☐ [techGetWidthSpacingSnapPatternDefName](#)
- ☐ [techGetWidthSpacingSnapPatternDefOffset](#)
- ☐ [techGetWidthSpacingSnapPatternDefPatternGroupNames](#)
- ☐ [techGetWidthSpacingSnapPatternDefPatternNames](#)
- ☐ [techGetWidthSpacingSnapPatternDefPeriod](#)
- ☐ [techGetWidthSpacingSnapPatternDefPurpose](#)
- ☐ [techGetWidthSpacingSnapPatternDefSnappingLayers](#)
- ☐ [techIsWidthSpacingSnapPatternDefHorizontal](#)

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

☐ [techStartGenWidthSpacingSnapPatternDefForLPP](#)

☐ [techStartGenWidthSpacingSnapPatternDef](#)

■ [Related Snap Pattern Functions](#)

☐ [techAllocRelatedSnapPatternsSpecArray](#)

☐ [techCreateRelatedSnapPatterns](#)

☐ [techDeleteRelatedSnapPatterns](#)

☐ [techFindRelatedSnapPatterns](#)

☐ [techFreeRelatedSnapPatternsSpecArray](#)

☐ [techGenRelatedSnapPatterns](#)

☐ [techGetRelatedSnapPatternsName](#)

☐ [techGetRelatedSnapPatternsSpec](#)

☐ [techStartGenRelatedSnapPatterns](#)

Miscellaneous Technology File Functions

techMaskColorTypeToString

```
String  
techMaskColorTypeToString(  
    int  type  
);
```

Description

Converts a `techColorMask` type enum to a string holding the name of the corresponding mask color type. Valid input values are integers from 0 to 3, or integer constants `techcGrayColorMask`, `techcMask1ColorMask`, `techcMask2ColorMask` and `techcMask3ColorMask`, respectively.

Arguments

<i>type</i>	A <code>techMaskColor</code> type enum.

Return Value

<i>String</i>	A mask color type name. Valid values are "grayColor", "mask1Color", "mask2Color" and "mask3Color".

techStringToMaskColorType

Boolean

```
techStringToMaskColorType(  
    constString      typeStr,  
    techColorMaskType *type  
);
```

Description

Converts the string holding a mask color type name to a `techColorMask` type enum. Valid input strings are "grayColor", "mask1Color", "mask2Color", and "mask3Color". The result is written to the memory area referenced by the `type` pointer.

Arguments

<code>typeStr</code>	A mask color type name.
<code>type</code>	Pointer to the memory area where the conversion result will be written. The value referenced by the pointer is undefined if <code>FALSE</code> is returned.

Return Value

<code>Boolean</code>	TRUE, if the conversion returned a valid result. FALSE, in case of failure.

Width Spacing Pattern Functions

techAllocWidthSpacingPatternSpecArray

```
techRelatedSnapPatternsSpec*  
techAllocWidthSpacingPatternSpecArray(  
    unsigned int numPatterns  
);
```

Description

Allocates memory for the array of width spacing pattern specifications and initializes the memory with zeros.

Arguments

<i>numPatterns</i>	Number of width spacing pattern specifications to fit in the allocated memory.

Return Value

<i>techRelatedSnapPatternsSpec*</i>	Pointer to the allocated memory area.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techCreateWidthSpacingPattern

techWidthSpacingPatternId

```
techCreateWidthSpacingPattern(  
    techFileId          techfile,  
    constString         name,  
    techWidthSpacingPatternSpec *patterns,  
    unsigned int        numPatterns,  
    dbDistance          offset,  
    Boolean             isRepeatOffset,  
    techColorMaskType   startingColor,  
    Boolean             isShiftColor  
);
```

Description

Creates a width spacing pattern in the specified technology file.

Arguments

<i>techfile</i>	ID of the technology file in which the width spacing pattern is created.
<i>name</i>	Name for the width spacing pattern.
<i>patterns</i>	Array of pattern specifications.
<i>numPatterns</i>	Number of pattern specifications in the array of patterns.
<i>offset</i>	Distance of the first track from the period track.
<i>isRepeatOffset</i>	Flag to indicate whether the offset is applied when a pattern repeats.
<i>startingColor</i>	Color of the first track. If the color of the first track is not gray, all tracks are automatically colored such that the color pattern changes on each track. In other words, you cannot specify the color for a track other than the first one or leave a track uncolored.
<i>isShiftColor</i>	Flag to indicate whether track colors change when a pattern repeats. This attribute is used only when all tracks are individually colored. If no color is specified for a track, it remains uncolored (that is, gray).

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

Return Value

techWidthSpacingPatternId

ID of the created width spacing pattern.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techDeleteWidthSpacingPattern

Boolean

```
techDeleteWidthSpacingPattern(  
    techWidthSpacingPatternId    wspId  
);
```

Description

Deletes the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techFindWidthSpacingPattern

```
techWidthSpacingPatternId  
techFindWidthSpacingPattern(  
    techFileId  techfile,  
    constString name  
);
```

Description

Searches for the width spacing pattern with the specified name in the specified technology file.

Arguments

<i>techfile</i>	ID of the technology file in which the search is performed.
<i>name</i>	Name of the width spacing pattern for which the search is performed.

Return Value

<i>techWidthSpacingPatternId</i>	ID of the width spacing pattern, if found. Returns <code>NULL</code> in case of failure.

techFreeWidthSpacingPatternSpecArray

```
void  
techFreeWidthSpacingPatternSpecArray(  
    unsigned int          numPatterns,  
    techWidthSpacingPatternSpec *patterns  
);
```

Description

Deallocates the memory allocated for an array of width spacing pattern specifications.

Note: In addition to deallocating the specified memory area, the function also frees the memory internally on the non-null `colors`, `wireTypes`, and `specs` members of each width spacing snap pattern specification in the array.

Arguments

<i>numPatterns</i>	Size of the deallocated memory area, measured in blocks of <code>sizeof(techWidthSpacingPatternSpec)</code> bytes.
<i>patterns</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGenWidthSpacingPattern

Boolean

```
techGenWidthSpacingPattern(  
    techGenStateId          pState,  
    techWidthSpacingPatternId *wspId  
);
```

Description

Retrieves the next width spacing pattern ID from the generator with the given `techGenStateId`. The widthSpacingPattern ID is returned in the `wspId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>relatedSPId</i>	ID of the next <code>widthSpacingPattern</code> produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if ID of the next width spacing pattern is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGenWidthSpacingPattern

Boolean

```
techGenWidthSpacingPattern(  
    techGenStateId          pState,  
    techWidthSpacingPatternId *wspId  
);
```

Description

Retrieves the next width spacing pattern ID from the generator with the given `techGenStateId`. The widthSpacingPattern ID is returned in the `wspId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>relatedSPId</i>	ID of the next <code>widthSpacingPattern</code> produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if ID of the next width spacing pattern is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

techGetWidthSpacingPatternAllowedRepeatMode

```
techWSPAllowedRepeatMode  
  
techGetWidthSpacingPatternAllowedRepeatMode (  
    techWidthSpacingPatternId    wspId  
);
```

Description

Returns the allowed repeat mode set for a width spacing pattern.

Arguments

<i>wspId</i>	ID of the width spacing pattern.

Return Value

<i>techWSPAllowedRepeatMode</i>	The allowed repeat mode set for the width spacing pattern.

techGetWidthSpacingPatternDefaultRepeatMode

```
techWSPRepeatMode  
  
techGetWidthSpacingPatternDefaultRepeatMode (  
    techWidthSpacingPatternId    wspId  
);
```

Description

Returns the default repeat mode set for a width spacing pattern.

Arguments

<i>wspId</i>	ID of the width spacing pattern.

Return Value

techWSPRepeatMode	The default repeat mode set for the width spacing pattern.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternName

String

```
techGetWidthSpacingPatternName (  
    techWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the name for the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>String</i>	Name of the specified width spacing pattern object. Returns <code>NULL</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternOffset

```
dbDistance  
techGetWidthSpacingPatternOffset(  
    techWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the offset, which is the distance of the first track from the period track, for the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>dbDistance</i>	The offset value of the specified width spacing pattern object. Returns 0 in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternPatterns

```
void
techGetWidthSpacingPatternPatterns (
    techWidthSpacingPatternId    wspId,
    unsigned int                  *numPatternSpecs,
    techWidthSpacingPatternSpec  **patternSpecs
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of pattern specifications of the specified width spacing pattern object. Pointer to the allocated memory area is returned through the *patternSpecs* parameter and the size of this memory area is returned through the *numPatternSpecs* parameter.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.
<i>numPatternSpecs</i>	Pointer to the memory area where the number of pattern specifications is placed.
<i>patternSpecs</i>	Pointer to the memory area where the array of pattern specifications is placed.

Return Value

No value returned.

techGetWidthSpacingPatternRepeatOffset

Boolean

```
techGetWidthSpacingPatternRepeatOffset(  
    techWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the repeat offset flag for the specified width spacing pattern object. The repeat offset flag is a Boolean value indicating whether the offset is applied when a pattern repeats.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>Boolean</i>	Repeat offset flag of the specified width spacing pattern object. Returns <code>FALSE</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternShiftColor

Boolean

```
techGetWidthSpacingPatternShiftColor(  
    techWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the shift color flag for the specified width spacing pattern object. The shift color flag is a Boolean value indicating whether the track colors shift when a pattern repeats.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>Boolean</i>	The shift color flag of the specified width spacing pattern object. Returns <code>FALSE</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternStartingColor

```
techColorMaskType  
techGetWidthSpacingPatternStartingColor(  
    techWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the color of the first track for the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>techColorMaskType</i>	Color of the first track. Returns <code>techcGrayColorMask</code> in case of error.

techSetWidthSpacingPatternRepeatMode

Boolean

```
techSetWidthSpacingPatternRepeatMode (  
    techWidthSpacingPatternId    wspId,  
    techWSPAllowedRepeatMode     allowedMode,  
    techWSPRepeatMode            defaultMode  
);
```

Description

Sets the allowed and default repeat modes for a width spacing pattern.

Arguments

<i>wspId</i>	ID of the width spacing pattern.
<i>allowedMode</i>	The allowed repeat pattern mode.
<i>defaultMode</i>	The default repeat pattern mode.

Return Value

Boolean	TRUE, if the allowed and default repeat modes were set. FALSE, if the allowed and default repeat modes were not set.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techStartGenWidthSpacingPattern

```
techGenStateId  
techStartGenWidthSpacingPattern(  
    techFileId      techId  
);
```

Description

Initializes a generator for all width spacing patterns in the specified technology file. A generator state ID is returned, which is used with the [techGenWidthSpacingPattern](#) function.

Arguments

<i>techId</i>	ID of the technology file containing the width spacing patterns to be generated.

Return Value

<i>techGenStateId</i>	State ID for the generator.

Width Spacing Pattern Group Functions

techCreateWidthSpacingPatternGroup

```
techWidthSpacingPatternGroupId  
techCreateWidthSpacingPatternGroup(  
    techFileId    techfile,  
    constString   name,  
    constString   *patternNames,  
    unsigned int   numPatternNames  
);
```

Description

Creates a group of width spacing patterns in the specified technology file.

Arguments

<i>techfile</i>	ID of the technology file in which the width spacing pattern group is created.
<i>name</i>	Name of the width spacing pattern group.
<i>patternNames</i>	Array of the width spacing pattern names to include in the group.
<i>numPatternNames</i>	Number of elements in the array of pattern names.

Return Value

<i>techWidthSpacingPatternGroupId</i>	ID of the newly created width spacing pattern group.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techDeleteWidthSpacingPatternGroup

Boolean

```
techDeleteWidthSpacingPatternGroup(  
    techWidthSpacingPatternGroupId    WSPGId  
);
```

Description

Deletes the specified width spacing pattern group object.

Arguments

<i>WSPGId</i>	ID of a width spacing pattern group object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techFindWidthSpacingPatternGroup

```
techWidthSpacingPatternGroupId  
techFindWidthSpacingPatternGroup(  
    techFileId  techfile,  
    constString name  
);
```

Description

Searches for the width spacing pattern group with the specified name in the specified technology file.

Arguments

<i>techfile</i>	ID of the technology file in which the search is performed.
<i>name</i>	Name for the width spacing pattern group for which the search is performed.

Return Value

<i>techWidthSpacingPatternGroupId</i>	ID of the width spacing pattern group, if found. Returns <code>NULL</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGenWidthSpacingPatternGroup

Boolean

```
techGenWidthSpacingPatternGroup(  
    techGenStateId          pState,  
    techWidthSpacingPatternGroupId *wspGroupId  
);
```

Description

Retrieves the ID of the next width spacing pattern group from the generator with the given *techGenStateId*. The *widthSpacingPatternGroup* ID is returned in the *wspGroupId* argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wspGroupId</i>	ID of the next width spacing pattern group produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	TRUE, if ID of the next width spacing pattern group is successfully generated. FALSE, if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternGroupName

String

```
techGetWidthSpacingPatternGroupName (  
    techWidthSpacingPatternGroupId    WSPGId  
);
```

Description

Retrieves the name for the specified width spacing pattern group object.

Arguments

<i>WSPGId</i>	ID of a width spacing pattern group object.

Return Value

<i>String</i>	Name of the specified width spacing pattern group object.
	Returns NULL in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingPatternGroupPatternNames

```
void
techGetWidthSpacingPatternGroupPatternNames (
    techWidthSpacingPatternGroupId  WSPGId,
    unsigned int                    *numPatterns,
    String                          **patterns
);
```

Description

Allocates a suitable amount of heap memory and fills it with the array of pattern names of the specified width spacing pattern group. Pointer to the allocated memory area is returned through the *patterns* parameter and the size of this memory area is returned through the *numPatterns* parameter.

Arguments

<i>WSPGId</i>	ID of a width spacing pattern group object.
<i>numPatterns</i>	Pointer to the memory area where the number of pattern names is placed.
<i>patterns</i>	Pointer to the memory area where the array of pattern names is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techStartGenWidthSpacingPatternGroup

```
techGenStateId  
techStartGenWidthSpacingPatternGroup(  
    techFileId      techId  
);
```

Description

Initializes a generator for all width spacing pattern groups in the specified technology file. A generator state ID is returned, which is used with [techGenWidthSpacingPatternGroup](#) the function.

Arguments

<i>techId</i>	ID of the technology file containing the width spacing pattern groups to be generated.

Return Value

<i>techGenStateId</i>	State ID for the generator.

Width Spacing Snap Pattern Definition Functions

techCreateWidthSpacingSnapPatternDef

```
techWidthSpacingSnapPatternDefId
techCreateWidthSpacingSnapPatternDef(
    techFileId          techId,
    constString          name,
    techLayerNum         layerNum,
    techPurpose          purposeNum,
    dbDistance           period,
    Boolean              isHorizontal,
    dbDistance           offset,
    techSnappingLayer    *snappingLayers,
    unsigned int         numSnappingLayers,
    String               *patternNames,
    unsigned int         numPatterns,
    String               *patternGroupNames,
    unsigned int         numPatternGroups,
    String               defaultActive
);
```

Description

Creates a width spacing snap pattern definition object in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the width spacing snap pattern definition object is created.
<i>name</i>	Name for the width spacing snap pattern definition object.
<i>layerNum</i>	Layer number for the width spacing snap pattern definition object.
<i>purposeNum</i>	Purpose number for the width spacing snap pattern definition object.
<i>period</i>	Spacing between coarse-grain period tracks (heart-beat period).
<i>isHorizontal</i>	Pattern direction: TRUE for horizontal and FALSE for vertical.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

<i>offset</i>	Distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).
<i>snappingLayers</i>	Array of snapping layers for the width spacing snap pattern definition object.
<i>numSnappingLayers</i>	Number of elements in the array of snapping layers.
<i>patternNames</i>	Array of allowed width spacing pattern names.
<i>numPatterns</i>	Number of elements in the array of pattern names.
<i>patternGroupNames</i>	Array of allowed width spacing pattern group names.
<i>numPatternGroups</i>	Number of elements in the array of pattern group names.
<i>defaultActive</i>	Pattern to be used in areas where no region has been drawn.

Return Value

<i>techWidthSpacingSnapPatternDefId</i>	ID of the newly created width spacing snap pattern definition object.

techDeleteWidthSpacingSnapPatternDef

```
Boolean  
techDeleteWidthSpacingSnapPatternDef(  
    techWidthSpacingSnapPatternDefId spDefId  
);
```

Description

Deletes the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted.
	FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techFindWidthSpacingSnapPatternDefByName

```
techWidthSpacingSnapPatternDefId  
techFindWidthSpacingSnapPatternDefByName (  
    techFileId      techId,  
    constString     name  
);
```

Description

Searches for the width spacing snap pattern definition object with the specified name in the specified technology file.

Arguments

<i>techId</i>	ID of the technology file in which the search is performed.
<i>name</i>	Name of the width spacing pattern for which the search is performed.

Return Value

<i>techWidthSpacingSnapPatternDefId</i>	ID of the width spacing snap pattern definition object, if found. Returns <code>NULL</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGenWidthSpacingSnapPatternDef

Boolean

```
techGenWidthSpacingSnapPatternDef(  
    techGenStateId                pState,  
    techWidthSpacingSnapPatternDefId *wsspDefId  
);
```

Description

Retrieves the ID of the next width spacing snap pattern definition from the generator with the given `techGenStateId`. The `widthSpacingSnapPatternDef` ID is returned in the `wsspDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wsspDefId</i>	ID of the next width spacing snap pattern definition produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE, if ID of the next width spacing snap pattern definition is successfully generated. FALSE, if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGenWidthSpacingSnapPatternDefForLPP

Boolean

```
techGenWidthSpacingSnapPatternDefForLPP(  
    techGenStateId          pState,  
    techWidthSpacingSnapPatternDefId *wsspDefId  
);
```

Description

Retrieves the ID of the next width spacing snap pattern definition object from the generator using the given `techGenStateId`. The `widthSpacingPatternDef` layer-purpose pair (LPP) ID is returned in the `wsspDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wsspDefId</i>	ID of the next width spacing snap pattern definition object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the ID of the next width spacing pattern definition object is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

techGetWidthSpacingSnapPatternDefDefaultActiveName

String

```
techGetWidthSpacingSnapPatternDefDefaultActiveName (  
    techWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the name of the default active pattern for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>String</i>	Default active pattern name for the specified width spacing snap pattern definition object. Returns <code>NULL</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingSnapPatternDefLayer

```
techLayerNum  
techGetWidthSpacingSnapPatternDefLayer(  
    techWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves layer number for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>techLayerNum</i>	Layer number for the specified width spacing snap pattern definition object. Returns <code>techcNullLayer</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingSnapPatternDefName

String

```
techGetWidthSpacingSnapPatternDefName (  
    techWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the name for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>String</i>	Name of the specified width spacing snap pattern definition object. Returns NULL in case of error.

techGetWidthSpacingSnapPatternDefOffset

```
dbDistance  
techGetWidthSpacingSnapPatternDefOffset(  
    techWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the offset for the specified width spacing snap pattern definition object. This is the distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>dbDistance</i>	Offset value of the specified width spacing snap pattern definition object. Returns 0 in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingSnapPatternDefPatternGroupNames

```
void
techGetWidthSpacingSnapPatternDefPatternGroupNames (
    techWidthSpacingSnapPatternDefId    spDefId,
    unsigned int                        *numPatternGroups,
    String                             **patternGroupNames
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of pattern group names of the specified width spacing snap pattern definition object. Pointer to the allocated memory area is returned through the *patternGroupNames* parameter and the size of this memory area is returned through the *numPatternGroups* parameter.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.
<i>numPatternGroups</i>	Pointer to the memory area where the number of pattern group names is placed.
<i>patternGroupNames</i>	Pointer to the memory area where the array of pattern group names is placed.

Return Value

No value returned.

techGetWidthSpacingSnapPatternDefPatternNames

```
void  
techGetWidthSpacingSnapPatternDefPatternNames (  
    techWidthSpacingSnapPatternDefId    spDefId,  
    unsigned int                        *numPatterns,  
    String                             **patternNames  
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of pattern names of the specified width spacing snap pattern definition object. Pointer to the allocated memory area is returned through the *patternNames* parameter and the size of this memory area is returned through the *numPatterns* parameter.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.
<i>numPatterns</i>	Pointer to the memory area where the number of pattern names is placed.
<i>patternNames</i>	Pointer to the memory area where the array of pattern names is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingSnapPatternDefPeriod

dbDistance

```
techGetWidthSpacingSnapPatternDefPeriod(  
    techWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the spacing between coarse-grain period tracks for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>dbDistance</i>	Space between coarse-grain period tracks. Returns 0 in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetWidthSpacingSnapPatternDefPurpose

techPurpose

```
techGetWidthSpacingSnapPatternDefPurpose(
    techWidthSpacingSnapPatternDefId    spDefId
);
```

Description

Retrieves the purpose for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>techPurpose</i>	The purpose number for the specified width spacing snap pattern definition object. Returns <code>techcUnknownPurpose</code> in case of error.

techGetWidthSpacingSnapPatternDefSnappingLayers

void

```
techGetWidthSpacingSnapPatternDefSnappingLayers (  
    techWidthSpacingSnapPatternDefId    spDefId,  
    unsigned int                        *numSnappingLayers,  
    techSnappingLayer                  **snappingLayers  
);
```

Description

Retrieves an array of snapping layers for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.
<i>numSnappingLayers</i>	Pointer to the memory area where the number of snapping layers is placed.
<i>snappingLayers</i>	Pointer to the memory area where the array of snapping layers is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techIsWidthSpacingSnapPatternDefHorizontal

Boolean

```
techIsWidthSpacingSnapPatternDefHorizontal(  
    techWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Verifies whether the specified width spacing snap pattern definition object is horizontal.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>Boolean</i>	TRUE, if the object is horizontal. FALSE, if the object is vertical.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techStartGenWidthSpacingSnapPatternDefForLPP

```
techGenStateId  
techStartGenWidthSpacingSnapPatternDefForLPP(  
    techFileId        techId,  
    techLayerId       layerId,  
    techPurposeId     purposeId  
);
```

Description

Initializes a generator for all the width spacing snap pattern definition objects in the technology file specified by the `techFileId` that have the given LPP. A generator state ID is returned, which is used with the `techGenWidthSpacingSnapPatternDefForLPP` function.

Arguments

<i>techId</i>	ID of the technology file containing the width spacing snap pattern definition objects to be generated.
<i>layerId</i>	Layer ID in a layer-purpose pair.
<i>purposeId</i>	Purpose ID in a layer-purpose pair.

Return Value

<i>techGenStateId</i>	State ID for the generator.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techStartGenWidthSpacingSnapPatternDef

```
techGenStateId  
techStartGenWidthSpacingSnapPatternDef(  
    techFileId          techId  
);
```

Description

Initializes a generator for all width spacing snap pattern definition objects in the specified technology file. A generator state ID is returned, which is used with the techGenWidthSpacingSnapPatternDef function.

Arguments

<i>techId</i>	ID of the technology file containing the width spacing snap pattern definitions to be generated.

Return Value

<i>techGenStateId</i>	State ID for the generator.

Related Snap Pattern Functions

techAllocRelatedSnapPatternsSpecArray

```
techRelatedSnapPatternsSpec*  
techAllocRelatedSnapPatternsSpecArray(  
    unsigned int    numSpecs  
);
```

Description

Allocates memory for the array of related snap patterns specifications and initializes the memory with zeroes.

Arguments

<i>numSpecs</i>	Number of related snap pattern specifications to fit in the allocated memory.

Return Value

<i>techRelatedSnapPatternsSpec*</i>	Pointer to the allocated memory area.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techCreateRelatedSnapPatterns

techRelatedSnapPatternsId

```
techCreateRelatedSnapPatterns (
    techFileId          techfile,
    constString         name,
    techRelatedSnapPatternsSpec *relatedSnapPatterns,
    unsigned int        numRelatedSnapPatterns
);
```

Description

Creates a group of related snap patterns in the specified technology file.

Arguments

<i>techfile</i>	ID of the technology file in which the related snap patterns are created.
<i>name</i>	Name for the group of related snap patterns.
<i>relatedSnapPatterns</i>	Array of related snap pattern definitions to include in the group.
<i>numRelatedSnapPatterns</i>	Number of elements in the array of related snap patterns.

Return Value

<i>techRelatedSnapPatternsId</i>	ID of the newly created group of related snap patterns.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techDeleteRelatedSnapPatterns

Boolean

```
techDeleteRelatedSnapPatterns(  
    techRelatedSnapPatternsId relatedSnapPatternsId  
);
```

Description

Deletes the specified related snap patterns object.

Arguments

<i>relatedSnapPatternsId</i>	ID of a related snap patterns object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techFindRelatedSnapPatterns

```
techRelatedSnapPatternsId  
techFindRelatedSnapPatterns(  
    techFileId      techfile,  
    constString     name  
);
```

Description

Searches for the group of related snap patterns with the specified name in the specified technology file.

Arguments

<i>techfile</i>	ID of the technology file in which the search is performed.
<i>name</i>	Name of the group of related snap patterns for which the search is performed.

Return Value

<i>techRelatedSnapPatternsId</i>	ID of the group of related snap patterns, if found. Returns <code>NULL</code> in case of failure.

techFreeRelatedSnapPatternsSpecArray

```
void  
techFreeRelatedSnapPatternsSpecArray(  
    unsigned int          numSpecs,  
    techRelatedSnapPatternsSpec *patternSpecs  
);
```

Description

Deallocates the memory allocated for an array of related snap patterns specifications.

Note: In addition to deallocating the specified memory area, the function also frees the memory internally on the non-null `patterns` and `patternGroups` members of each related snap patterns specification in the array.

Arguments

<i>numSpecs</i>	Size of the deallocated memory area, measured in blocks of <code>sizeof(techRelatedSnapPatternsSpec)</code> bytes.
<i>patternSpecs</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGenRelatedSnapPatterns

Boolean

```
techGenRelatedSnapPatterns(  
    techGenStateId      pState,  
    techRelatedSnapPatternsId  *relatedSPId  
);
```

Description

Retrieves the ID of the next related snap patterns from the generator with the given `techGenStateId`. The `relatedSnapPatterns` ID is returned in the *relatedSPId* argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>relatedSPId</i>	ID of the next related snap pattern produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if ID of the next related snap patterns is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

techGetRelatedSnapPatternsName

String

```
techGetRelatedSnapPatternsName (  
    techRelatedSnapPatternsId    relatedSnapPatternsId  
);
```

Description

Retrieves the name for the specified related snap patterns object.

Arguments

<i>relatedSnapPatternsId</i>	ID of a related snap patterns object.

Return Value

<i>String</i>	Name of the specified related snap patterns object. Returns <code>NULL</code> in case of error.

techGetRelatedSnapPatternsSpec

```
void  
techGetRelatedSnapPatternsSpec (  
    techRelatedSnapPatternsId    relatedSnapPatternsId,  
    unsigned int                  *numSpecs,  
    techRelatedSnapPatternsSpec **patternSpecs  
);
```

Description

Retrieves an array of snap pattern specifications for the specified related snap patterns object.

Allocates a suitable amount of heap memory, and fills it with the array of related snap pattern specifications of the specified related snap patterns object. Pointer to the allocated memory area is returned through the *patternSpecs* parameter and the size of this memory area is returned via the *numSpecs* parameter.

Arguments

<i>relatedSnapPatternsId</i>	ID of a related snap patterns object.
<i>numSpecs</i>	Pointer to the memory area where the number of snap pattern specifications is placed.
<i>patternSpecs</i>	Pointer to the memory area where the array of snap pattern specifications is placed.

Return Value

No value returned.

techStartGenRelatedSnapPatterns

```
techGenStateId  
techStartGenRelatedSnapPatterns (  
    techFileId          techId  
);
```

Description

Initializes a generator for all related snap patterns in the specified technology file. A generator state ID is returned, which is used with the [techGenRelatedSnapPatterns](#) function.

Arguments

<i>techId</i>	ID of the technology file containing the related snap patterns to be generated.

Return Value

<i>techGenStateId</i>	State ID for the generator.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

Example- Using the WSP Technology File Functions

```
// Create a scratch technology database.
ddId                      libId = ddCreateLib("design",
                                              getTestDir() + "/design");

techFileId                techfile = techOpenTechFile("design", "tech.db",
                                              "a");

// *** WIDTH SPACING PATTERN API EXAMPLES ***

// Create the test data.
const unsigned int        num_wsps = 3;
constString                *wspNames = calloc(num_wsps,
                                              constString);

wspNames[0] = strdup("1X");
wspNames[1] = strdup("2X");
wspNames[2] = strdup("1X2X");

const unsigned int        num_colors = 2;
techColorMaskType         *colors = calloc(num_colors,
                                              techColorMaskType);

colors[0] = techcGrayColorMask;

// Alternative way to set a mask color.
techStringToMaskColorType("mask3Color", &colors[1]);

const unsigned int        num_wireTypes = 2;
constString                *wireTypes = calloc(num_wireTypes,
                                              constString);

wireTypes[0] = strdup("wire0");
wireTypes[1] = strdup("wire1");

const unsigned int        num_baseSpecs = 2;
techWidthSpacingSpec       *baseSpecs = calloc(num_baseSpecs,
                                              techWidthSpacingSpec);

baseSpecs[0] = { 20, 20, techcGrayColorMask, (String) wireTypes[0] };
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
baseSpecs[1] = { 10, 10, techcGrayColorMask, (String) wireTypes[1] };

const unsigned int          num_wspSpecs = 2;
techWidthSpacingPatternSpec *wspSpecs
    = techAllocWidthSpacingPatternSpecArray(num_wspSpecs);

// Note: techFreeWidthSpacingPatternSpecArray() deallocates all members of the
// array and assumes that all members were allocated on the heap. This means
// that techWidthSpacingPatternSpec members must be allocated with a heap
// allocation function such as calloc().

wspSpecs[0] = {3, NULL, 0, NULL, 0, baseSpecs, num_baseSpecs };
wspSpecs[1] = {3, colors, num_colors, (String *) wireTypes, num_wireTypes,
               NULL, 0 };

const unsigned int          num_wspSpecs_invalid = 2;
techWidthSpacingPatternSpec *wspSpecs_invalid
    = techAllocWidthSpacingPatternSpecArray(num_wspSpecs_invalid);

techColorMaskType          *colors2 = calloc(num_colors,
                                              techColorMaskType);
memcpy(colors2, colors, num_colors * sizeof(techColorMaskType));

constString                *wireTypes2 = calloc(num_wireTypes,
                                                  constString);
memcpy(wireTypes2, wireTypes, num_wireTypes * sizeof(constString));

techWidthSpacingSpec       *baseSpecs2 = calloc(num_baseSpecs,
                                                  techWidthSpacingSpec);
techWidthSpacingSpec       *baseSpecs3 = calloc(num_baseSpecs,
                                                  techWidthSpacingSpec);

memcpy(baseSpecs2, baseSpecs, num_baseSpecs * sizeof(techWidthSpacingSpec));
memcpy(baseSpecs3, baseSpecs, num_baseSpecs * sizeof(techWidthSpacingSpec));

// This spec is invalid because it specifies the list of repeat colors
// both in the widthSpacingSpec and in the widthSpacingPattern spec.
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
wspSpecs_invalid[0] = {3, colors2, num_colors, NULL, 0,
                      baseSpecs2, num_baseSpecs };

// This spec is invalid because it specifies the list of wire types
// both in the widthSpacingSpec and in the widthSpacingPattern spec.

wspSpecs_invalid[1] = {3, NULL, 0, (String *) wireTypes2, num_wireTypes,
                      baseSpecs3, num_baseSpecs };

// These commands will fail due to the invalid specs.

techCreateWidthSpacingPattern(techfile, "bad_wsp1", wspSpecs_invalid, 1,
                             0, FALSE, techcGrayColorMask, FALSE);

techCreateWidthSpacingPattern(techfile, "bad_wsp2", wspSpecs_invalid + 1, 1,
                             0, FALSE, techcGrayColorMask, FALSE);

// This command will fail because we cannot define both the startingColor
// parameter and the list of repeat colors in the specification.

techCreateWidthSpacingPattern(techfile, "bad_wsp3", wspSpecs, 2,
                             0, FALSE, techcMask3ColorMask, FALSE);

// Generate valid widthSpacingPatterns.

techWidthSpacingPatternId wsps[num_wsps];

for (int i = 0; i < num_wsps; i++) {
    wsps[i] = techCreateWidthSpacingPattern(techfile, wspNames[i], wspSpecs,
                                           num_wspSpecs, 0, FALSE,
                                           techcGrayColorMask, FALSE);

    assert(wsps[i]);
}

// Examples for the getter functions.
cout << "Sample WSP dump: " << endl;
cout << "  name: " << techGetWidthSpacingPatternName(wsps[0]) << endl;
cout << "  offset: " << techGetWidthSpacingPatternOffset(wsps[0]) << endl;
cout << "  repeat offset: " << techGetWidthSpacingPatternRepeatOffset(wsps[0])
    << endl;
cout << "  shift color: " << (bool) techGetWidthSpacingPatternShiftColor(wsps[0])
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
<< endl;
cout << "    starting color: "
    <<
techMaskColorTypeToString(techGetWidthSpacingPatternStartingColor(wsps[0])) <<
endl;
cout << endl << endl;

unsigned int                tmp_num = 0;
techWidthSpacingPatternSpec *tmp_specs = NULL;

techGetWidthSpacingPatternPatterns(wsps[0], &tmp_num, &tmp_specs);
assert(tmp_num == num_wspSpecs);
techFreeWidthSpacingPatternSpecArray(tmp_num, tmp_specs);

// Examples for the generator functions.

techGenStateId              gen_state = techStartGenWidthSpacingPattern(techfile);
techWidthSpacingPatternId tmp_wsp;

cout << "Pattern names returned by the generator: ";

while (techGenWidthSpacingPattern(gen_state, &tmp_wsp))
    cout << " " << techGetWidthSpacingPatternName(tmp_wsp);

cout << endl << endl;
techStopGen(gen_state);

// Examples for the find and delete functions.

assert(techFindWidthSpacingPattern(techfile, "tmp_wsp")
    == techcNullWidthSpacingPatternId);
tmp_wsp = techCreateWidthSpacingPattern(techfile, "tmp_wsp", wspSpecs,
    num_wspSpecs, 0, FALSE,
    techcGrayColorMask, FALSE);
assert(techFindWidthSpacingPattern(techfile, "tmp_wsp"));
assert(techDeleteWidthSpacingPattern(tmp_wsp));
assert(techFindWidthSpacingPattern(techfile, "tmp_wsp")
    == techcNullWidthSpacingPatternId);
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
// *** WIDTH SPACING PATTERN GROUP API EXAMPLES ***

constString          wspgNames[] = {"wspg1"};
const unsigned int   num_wspgs = ARRAY_SIZE(wspgNames);

// This command will fail because a nonexisting widthSpacingPattern name
// is specified.
constString          bad_wsp = "bad_wsp";

techCreateWidthSpacingPatternGroup(techfile, NULL, &bad_wsp, 1);

// Generate valid widthSpacingPatternGroups.
techWidthSpacingPatternGroupId   wspgs[num_wspgs];

for (int i = 0; i < num_wspgs; i++) {
    wspgs[i] = techCreateWidthSpacingPatternGroup(techfile, wspgNames[i],
                                                    &wspNames[0],
                                                    num_wspgs);

    assert(wspgs[i]);
}

// Examples for the getter functions.
cout << "Sample WSP group dump: " << endl;
cout << "  name: " << techGetWidthSpacingPatternGroupName(wspgs[0]) << endl;
cout << "  patterns:";

constString*  tmp_names = NULL;

techGetWidthSpacingPatternGroupPatternNames(wspgs[0], &tmp_num,
                                              (String **) &tmp_names);

assert(tmp_num == num_wspgs);

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl << endl;

free(tmp_names);
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
// Examples for the generator functions.
gen_state = techStartGenWidthSpacingPatternGroup(techfile);
techWidthSpacingPatternGroupId tmp_wspg;

cout << "Pattern group names returned by the generator: ";

while (techGenWidthSpacingPatternGroup(gen_state, &tmp_wspg))
    cout << " " << techGetWidthSpacingPatternGroupName(tmp_wspg);

cout << endl << endl;
techStopGen(gen_state);

// Examples for the find and delete functions.
assert(techFindWidthSpacingPatternGroup(techfile, "tmp_wspg")
    == techcNullWidthSpacingPatternGroupId);
tmp_wspg = techCreateWidthSpacingPatternGroup(techfile, "tmp_wspg",
    &wspNames[0], 1);
assert(techFindWidthSpacingPatternGroup(techfile, "tmp_wspg"));
assert(techDeleteWidthSpacingPatternGroup(tmp_wspg));
assert(techFindWidthSpacingPatternGroup(techfile, "tmp_wspg")
    == techcNullWidthSpacingPatternGroupId);

// *** WIDTH SPACING PATTERN DEF API EXAMPLES ***

Create the test data.

techLayerNum      layers[] = { 6, 8 };
techPurpose       purposes[] = { 1001, 1002 };
techLayerNum      snapLayerNum = 8;
techPurpose       snapPurposeNum = 5;
Boolean           isHorizontal = TRUE;
dbDistance        offset = 30;
dbDistance        period = 20;

const unsigned int num_snapLayers = 1;
techSnappingLayer *snapLayers = techAllocSnappingLayerArray(num_snapLayers);

snapLayers[0].layerNum = snapLayerNum;
```


Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
snapLayers[0].purposes = calloc(1, techPurpose);
snapLayers[0].numPurposes = 1;
snapLayers[0].purposes[0] = snapPurposeNum;

constString                                wsspDefNames[] = { "wsspdef1", "wsspdef2" };
const unsigned int                        num_wsspDefs = ARRAY_SIZE(wsspDefNames);
techWidthSpacingSnapPatternDefId          wsspDefs[num_wsspDefs];

constString                                defaultWSP = "1X2X";

for (int i = 0; i < num_wsspDefs; i++) {
    wsspDefs[i] = techCreateWidthSpacingSnapPatternDef(techfile,
                                                        wsspDefNames[i],
                                                        layers[i], purposes[i],
                                                        period, isHorizontal,
                                                        offset, snapLayers, 1,
                                                        &wspNames[0], num_wsps,
                                                        &wspgNames[0], num_wspgs,
                                                        defaultWSP);
    assert(wsspDefs[i]);
}

// Examples for the getter functions.
cout << "Sample widthSpacingSnapPatternDef dump: " << endl;
cout << "  name: "
    << techGetWidthSpacingSnapPatternDefName(wsspDefs[0]) << endl;
cout << "  layer: "
    << techGetWidthSpacingSnapPatternDefLayer(wsspDefs[0]) << endl;
cout << "  purpose: "
    << techGetWidthSpacingSnapPatternDefPurpose(wsspDefs[0]) << endl;
cout << "  period: "
    << techGetWidthSpacingSnapPatternDefPeriod(wsspDefs[0]) << endl;
cout << "  offset: "
    << techGetWidthSpacingSnapPatternDefOffset(wsspDefs[0]) << endl;
cout << "  horizontal: "
    << (bool) techIsWidthSpacingSnapPatternDefHorizontal(wsspDefs[0]) << endl;
cout << "  default active pattern: "
    << techGetWidthSpacingSnapPatternDefDefaultActiveName(wsspDefs[0])
    << endl;
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
techSnappingLayer*    tmp_snapLayers = NULL;

// Test techGetWidthSpacingSnapPatternDefSnappingLayers().
techGetWidthSpacingSnapPatternDefSnappingLayers(wsspDefs[0], &tmp_num,
                                                &tmp_snapLayers);

assert(tmp_num == num_snapLayers);
cout << "    snapping layers:" << endl;

for (int i = 0; i < tmp_num; i++) {
    cout << "        layer: " << tmp_snapLayers[i].layerNum << " purposes:";

    for (int j = 0; j < tmp_snapLayers[i].numPurposes; j++)
        cout << " " << tmp_snapLayers[i].purposes[j];

    cout << endl;
}

cout << endl;
techFreeSnappingLayerArray(tmp_num, tmp_snapLayers);

// Test techGetWidthSpacingSnapPatternDefPatternNames().
techGetWidthSpacingSnapPatternDefPatternNames(wsspDefs[0], &tmp_num,
                                              &tmp_names);

assert(tmp_num == num_wsps);
cout << "    patterns:";

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);

// Test techGetWidthSpacingSnapPatternDefPatternGroupNames().
techGetWidthSpacingSnapPatternDefPatternGroupNames(wsspDefs[0], &tmp_num,
                                                  &tmp_names);

assert(tmp_num == num_wspgs);
cout << "    pattern groups:";

for (int i = 0; i < tmp_num; i++)
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);

// Examples for the generator functions.
gen_state = techStartGenWidthSpacingSnapPatternDef(techfile);
techWidthSpacingSnapPatternDefId tmp_wsspdef;

cout << "Snap pattern definition names returned by the generator: ";

while (techGenWidthSpacingSnapPatternDef(gen_state, &tmp_wsspdef))
    cout << " " << techGetWidthSpacingSnapPatternDefName(tmp_wsspdef);

cout << endl << endl;
techStopGen(gen_state);

// Examples for the find and delete functions.
techLayerNum    tmp_layer = 10;
techPurpose     tmp_purpose = 1003;

assert(techFindWidthSpacingSnapPatternDefByName(techfile, "tmp_wsspdef")
       == techcNullWidthSpacingSnapPatternDefId);
tmp_wsspdef = techCreateWidthSpacingSnapPatternDef(techfile,
                                                    "tmp_wsspdef",
                                                    tmp_layer, tmp_purpose,
                                                    period, isHorizontal,
                                                    offset, snapLayers, 1,
                                                    NULL, 0,
                                                    NULL, 0,
                                                    defaultWSP);
assert(techFindWidthSpacingSnapPatternDefByName(techfile, "tmp_wsspdef"));
assert(techDeleteWidthSpacingSnapPatternDef(tmp_wsspdef));
assert(techFindWidthSpacingSnapPatternDefByName(techfile, "tmp_wsspdef")
       == techcNullWidthSpacingSnapPatternDefId);

tmp_wsspdef = techCreateWidthSpacingSnapPatternDef(techfile,
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
        "tmp_wsspdef",
        tmp_layer, tmp_purpose,
        period, isHorizontal,
        offset, snapLayers, 1,
        NULL, 0,
        NULL, 0,
        defaultWSP);

assert(techDeleteWidthSpacingSnapPatternDef(tmp_wsspdef));

// *** RELATED SNAP PATTERNS API EXAMPLES ***

// Create the test data.
const unsigned int      num_rspSpecs = 1;
techRelatedSnapPatternsSpec *rspSpecs =
techAllocRelatedSnapPatternsSpecArray(num_rspSpecs);

constString             *wspNames2 = calloc(num_wsps, constString);
constString             *wspgNames2 = calloc(num_wspgs, constString);

memcpy(wspNames2, wspNames, num_wsps * sizeof(constString));
memcpy(wspgNames2, wspgNames, num_wspgs * sizeof(constString));

rspSpecs[0] = { (String) wsspDefNames[0], (String *) wspNames2, num_wsps,
                (String *) wspgNames2, num_wspgs };

// Create invalid specs.
const unsigned int      num_rspSpecs_invalid = 3;
techRelatedSnapPatternsSpec *rspSpecs_invalid =
techAllocRelatedSnapPatternsSpecArray(num_rspSpecs_invalid);
constString             *wspNames_invalid = calloc(1,
                                                    constString);

constString             *wspgNames_invalid = calloc(1,
                                                    constString);

wspNames_invalid[0] = "bad_wsp";
wspgNames_invalid[0] = "bad_wspg";

// Nonexisting widthSpacingPattern name.
rspSpecs_invalid[0] = { (String) wsspDefNames[0],
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
        (String *) wspNames_invalid, 1, NULL, 0 };
// Nonexisting widthSpacingPatternGroup name.
rspSpecs_invalid[1] = { (String) wsspDefNames[0], NULL, 0,
        (String *) wspgNames_invalid, 1};
// Nonexisting widthSpacingSnapPatternDef name.
rspSpecs_invalid[2] = { "bad_wsspdef", NULL, 0, NULL, 0};

// These commands will fail due to invalid relatedSnapPatternsSpecs.
for (int i = 0; i < num_rspSpecs_invalid; i++)
    assert(techCreateRelatedSnapPatterns(techfile, "bad_rsp",
        rspSpecs_invalid + i, 1)
        == techcNullRelatedSnapPatternsId);

// This command generates a valid relatedSnapPatterns object.
techRelatedSnapPatternsId rsp = techCreateRelatedSnapPatterns(techfile,
        "rsp0",
        rspSpecs,
        num_rspSpecs);

assert(rsp);

// End of the examples section.
// The remaining auxiliary objects should be deallocated.
techFreeSnappingLayerArray(1, snapLayers);
techFreeWidthSpacingPatternSpecArray(num_wspSpecs, wspSpecs);
techFreeWidthSpacingPatternSpecArray(num_wspSpecs_invalid, wspSpecs_invalid);
techFreeRelatedSnapPatternsSpecArray(num_rspSpecs, rspSpecs);
techFreeRelatedSnapPatternsSpecArray(num_rspSpecs_invalid, rspSpecs_invalid);

return TRUE;
```

WSP Database Functions

Width Spacing Pattern (WSP) database functions are used to operate with `widthSpacingPattern`, `widthSpacingPatternGroup`, `patternRegion`, `widthSpacingSnapPatternDef`, and `relatedSnapPatterns` objects in the database.

These functions have the ability to create, find, and delete objects, access object parameters, and iterate over the objects stored in the database.

To learn how to use WSP database functions, see [Example- Using WSP Database Functions](#).

Supported functions include:

- Miscellaneous Database File Functions
 - ❑ [`dbMaskColorTypeToString`](#)
 - ❑ [`dbStringToMaskColorType`](#)
- Global Grid Offset Functions
 - ❑ [`dbGetGlobalGridDefaultRepeatMode`](#)
 - ❑ [`dbSetGlobalGridOffsetReferenceType`](#)
 - ❑ [`dbGetGlobalGridRepeatMode`](#)
 - ❑ [`dbSetGlobalGridDefaultRepeatMode`](#)
 - ❑ [`dbSetGlobalGridOffsetReferenceType`](#)
- Width Spacing Pattern Functions
 - ❑ [`dbAllocWidthSpacingPatternSpecArray`](#)
 - ❑ [`dbCreateWidthSpacingPattern`](#)
 - ❑ [`dbCreateWidthSpacingPatternWithRepeatMode`](#)
 - ❑ [`dbFindWidthSpacingPattern`](#)
 - ❑ [`dbFreeWidthSpacingPatternSpecArray`](#)
 - ❑ [`dbGenWidthSpacingPattern`](#)
 - ❑ [`dbGetWidthSpacingPatternAllowedRepeatMode`](#)
 - ❑ [`dbGetWidthSpacingPatternDefaultRepeatMode`](#)
 - ❑ [`dbGetWidthSpacingPatternName`](#)

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

- ☐ [dbGetWidthSpacingPatternOffset](#)
- ☐ [dbGetWidthSpacingPatternPatterns](#)
- ☐ [dbGetWidthSpacingPatternRepeatOffset](#)
- ☐ [dbGetWidthSpacingPatternShiftColor](#)
- ☐ [dbGetWidthSpacingPatternStartingColor](#)
- ☐ [dbStartGenWidthSpacingPattern](#)

■ [Width Spacing Pattern Group Functions](#)

- ☐ [dbCreateWidthSpacingPatternGroup](#)
- ☐ [dbDeleteWidthSpacingPatternGroup](#)
- ☐ [dbFindWidthSpacingPatternGroup](#)
- ☐ [dbGenWidthSpacingPatternGroup](#)
- ☐ [dbGetWidthSpacingPatternGroupName](#)
- ☐ [dbGetWidthSpacingPatternGroupPatternNames](#)
- ☐ [dbStartGenWidthSpacingPatternGroup](#)

■ [Pattern Region Functions](#)

- ☐ [dbGetPatternRegionCommittedPatterns](#)
- ☐ [dbGetPatternRegionActivePattern](#)
- ☐ [dbGetPatternRegionAllowedPatternGroups](#)
- ☐ [dbGetPatternRegionAllowedPatterns](#)
- ☐ [dbGetPatternRegionCreateType](#)
- ☐ [dbGetPatternRegionDefaultRepeatMode](#)
- ☐ [dbGetPatternRegionRepeatMode](#)
- ☐ [dbGetPatternRegionShiftColor](#)
- ☐ [dbHasPatternRegionActivePattern](#)
- ☐ [dbSetPatternRegionActivePattern](#)
- ☐ [dbSetPatternRegionAllowedPatternGroups](#)
- ☐ [dbSetPatternRegionAllowedPatterns](#)

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

- ☐ [dbSetPatternRegionRepeatMode](#)

- ☐ [dbShapeHasPatternRegionInfo](#)

- [Width Spacing Snap Pattern Definition Functions](#)

- ☐ [dbAllocSnappingLayerArray](#)

- ☐ [dbFreeSnappingLayerArray](#)

- ☐ [dbCreateWidthSpacingSnapPatternDef](#)

- ☐ [dbDeleteWidthSpacingSnapPatternDef](#)

- ☐ [dbFindWidthSpacingSnapPatternDefByName](#)

- ☐ [dbFindWidthSpacingSnapPatternDefByName](#)

- ☐ [dbGenWidthSpacingSnapPatternDef](#)

- ☐ [dbGenWidthSpacingSnapPatternDefForSnappingLPP](#)

- ☐ [dbGetWidthSpacingSnapPatternDefDefaultActiveName](#)

- ☐ [dbGetWidthSpacingSnapPatternDefLayer](#)

- ☐ [dbGetWidthSpacingSnapPatternDefName](#)

- ☐ [dbGetWidthSpacingSnapPatternDefOffset](#)

- ☐ [dbGetWidthSpacingSnapPatternDefPatternGroupNames](#)

- ☐ [dbGetWidthSpacingSnapPatternDefPatternNames](#)

- ☐ [dbGetWidthSpacingSnapPatternDefPeriod](#)

- ☐ [dbGetWidthSpacingSnapPatternDefPurpose](#)

- ☐ [dbGetWidthSpacingSnapPatternDefSnappingLayers](#)

- ☐ [dbIsWidthSpacingSnapPatternDefHorizontal](#)

- ☐ [dbStartGenWidthSpacingSnapPatternDef](#)

- ☐ [dbStartGenWidthSpacingSnapPatternDefForSnappingLPP](#)

- [Related Snap Pattern Functions](#)

- ☐ [dbAllocRelatedSnapPatternsSpecArray](#)

- ☐ [dbCreateRelatedSnapPatterns](#)

- ☐ [dbDeleteRelatedSnapPatterns](#)

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

- ❑ [dbFindRelatedSnapPatterns](#)
- ❑ [dbFreeRelatedSnapPatternsSpecArray](#)
- ❑ [dbGenRelatedSnapPatterns](#)
- ❑ [dbGetRelatedSnapPatternsName](#)
- ❑ [dbGetRelatedSnapPatternsSpec](#)
- ❑ [dbStartGenRelatedSnapPatterns](#)

Miscellaneous Database File Functions

dbMaskColorTypeToString

String

```
dbMaskColorTypeToString(  
    dbMaskColorType type  
);
```

Description

Converts a `dbMaskColorType` enum to a string holding the name of the corresponding mask color type. Valid input values are integers from 1 to 6, or integer constants `dbcGrayColor`, `dbcMask1Color`, `dbcMask2Color`, `dbcMask3Color`, `dbcBlackColor`, `dbcMultiColor`, respectively. If the input parameter is `dbcUnknownColor` (equivalent to 0) or any other integer outside the range 1-6, the output string is "unknown".

Arguments

<i>type</i>	A <code>dbMaskColorType</code> enum.

Return Value

<i>String</i>	A mask color type name. Valid values are "grayColor", "mask1Color", "mask2Color", "mask3Color", "blackColor", and "multiColor". Returns "unknown" in case of error.

dbStringToMaskColorType

```
dbMaskColorType  
dbStringToMaskColorType(  
    constString    s,  
);
```

Description

Converts the string holding a mask color type name to a `dbMaskColorType` enum. Valid input strings are "grayColor", "mask1Color", "mask2Color", "mask3Color", "blackColor" and "multiColor". Any other input string results in returning 0, which is same as `dbcUnknownColor`.

Arguments

<i>s</i>	A mask color type name.

Return Value

<i>dbMaskColorType</i>	A mask color enum value. Returns 0 or <code>dbcUnknownColor</code> in case of error.

Global Grid Offset Functions

dbGetGlobalGridDefaultRepeatMode

```
dbWSPRepeatMode  
dbGetGlobalGridDefaultRepeatMode(  
    dbCellViewId    cvId,  
    constString      defName  
);
```

Description

Returns the default repeat mode for the specified widthSpacingSnapPatternDef global grid in the specified cellview.

Arguments

<i>cvId</i>	ID of a cellview.
<i>defName</i>	Name of a widthSpacingSnapPatternDef object.

Return Value

<i>dbWSPRepeatMode</i>	Value of the default repeat mode enum value.

Example

```
dbGetGlobalGridDefaultRepeatMode(cvId, "defName")
```

Returns the default repeat mode enum value for the *defName* global grid in the cellview *cvId*.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetGlobalGridOffsetReferenceType

```
dbGlobalGridOffsetType  
dbGetGlobalGridOffsetReferenceType(  
    dbCellViewId cellViewId  
);
```

Description

Returns the global grid offset reference type of a cellview for snapPatternDef and widthSpacingSnapPatternDef global grids.

Arguments

<i>cellViewId</i>	ID of the cellview used to obtain the global grid offset reference.

Return Value

<i>dbGlobalGridOffsetType</i>	The value of the global grid offset reference. Valid Values: dbcBoundaryOffset or dbcOriginOffset.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetGlobalGridRepeatMode

```
dbWSPRepeatMode  
  
dbGetGlobalGridRepeatMode(  
    dbCellViewId    cvId,  
    constString      defName,  
    constString      wspName  
);
```

Description

Returns the repeat mode for the specified widthSpacingSnapPatternDef global grid in the specified cellview for the given width spacing pattern.

Arguments

<i>cvId</i>	ID of a cellview.
<i>defName</i>	Name of a widthSpacingSnapPatternDef object.
<i>wspName</i>	Name of a width spacing pattern.

Return Value

<i>dbWSPRepeatMode</i>	Value of a valid repeat mode enum value.

Example

```
dbGetGlobalGridRepeatMode(cvId, "defName", "wspName")
```

Returns the repeat mode enum value for the *defName* global grid in the cellview *cvId* for the width spacing pattern *wspName*.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetGlobalGridDefaultRepeatMode

Boolean

```
dbSetGlobalGridDefaultRepeatMode (
    dbCellViewId      cvId,
    constString        defName,
    dbWSPRepeatMode    repeatMode
);
```

Description

Sets the specified repeat mode as the default on a cellview for the specified widthSpacingSnapPatternDef global grid.

Arguments

<i>cvId</i>	ID of a cellview.
<i>defName</i>	Name of a widthSpacingSnapPatternDef object.
<i>repeatMode</i>	Name of the repeat mode enum value to be set. Valid values: dbcWSPNoRepeat, dbcWSPStepped, dbcWSPFlippedStartsWithOdd, dbcWSPFlippedStartsWithEven

Return Value

<i>Boolean</i>	TRUE, if the default repeat mode is successfully set for the specified widthSpacingSnapPatternDef global grid. FALSE, in case of failure.

Example

```
dbSetGlobalGridDefaultRepeatMode (cvId, "defName", dbcWSPStepped)
```

The value dbcWSPStepped is set as the default repeat mode enum value for the defName global grid in the cellview cvId.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetGlobalGridOffsetReferenceType

Boolean

```
dbSetGlobalGridOffsetReferenceType (
    dbCellViewId cellViewId,
    dbGlobalGridOffsetType type
);
```

Description

Sets the global grid offset reference on a cellview for snapPatternDef and widthSpacingSnapPatternDef global grids. The global grid reference can either be dbcBoundaryOffset or dbcOriginOffset.

Arguments

<i>cellViewId</i>	ID of the cellview used to store the global grid offset reference.
<i>type</i>	Value of the global grid reference. Valid values: dbcBoundaryOffset or dbcOriginOffset.

Return Value

<i>Boolean</i>	TRUE, if the global grid offset reference is set on the specified cellview. FALSE, in case of failure.

Width Spacing Pattern Functions

dbAllocWidthSpacingPatternSpecArray

```
dbWidthSpacingPatternSpec*  
dbAllocWidthSpacingPatternSpecArray(  
    unsigned int    numPatternSpecs  
);
```

Description

Allocates memory for the array of width spacing pattern specifications and initializes the memory with zeroes.

Arguments

<i>numPatternSpecs</i>	Number of width spacing pattern specifications to fit in the allocated memory

Return Value

<i>dbWidthSpacingPatternSpec*</i>	Pointer to the allocated memory area.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbCreateWidthSpacingPattern

dbWidthSpacingPatternId

```
dbCreateWidthSpacingPattern(  
    dbCellViewId          cv,  
    constString            name,  
    dbWidthSpacingPatternSpec *patterns,  
    unsigned int           numPatterns,  
    dbDistance             offset,  
    Boolean                isRepeatOffset,  
    dbMaskColorType        startingColor,  
    Boolean                isShiftColor  
);
```

Description

Creates a width spacing pattern in the specified cellview.

Arguments

<i>cv</i>	ID of the cellview in which the width spacing pattern is created.
<i>name</i>	Name for the width spacing pattern.
<i>patterns</i>	An array of pattern specifications.
<i>numPatterns</i>	Number of pattern specifications in the array of patterns.
<i>offset</i>	Distance of the first track from the period track.
<i>isRepeatOffset</i>	Flag to indicate whether the offset is applied when a pattern repeats.
<i>startingColor</i>	Color of the first track. If the color of the first track is not gray, all tracks are automatically colored such that the color pattern changes on each track. In other words, you cannot specify the color for a track other than the first one or leave a track uncolored.
<i>isShiftColor</i>	Flag to indicate whether track colors change when a pattern repeats. This attribute is used only when all tracks are individually colored. If no color is specified for a track, it remains uncolored (that is, gray).

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

<i>startingColor</i>	Color of the first track. If the color of the first track is not gray, all tracks are automatically colored so that the color shifts on each track, that is there is no way to specify color for the tracks other than the first one or to leave some tracks uncolored.
<i>isShiftColor</i>	Flag to indicate whether track colors shift when a pattern repeats. This attribute is used only when all tracks are individually colored and no automatic color assignment is done. If no color is specified for a track, then it would be uncolored (that is gray).

Return Value

dbWidthSpacingPatternId ID of the created width spacing pattern.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbCreateWidthSpacingPatternWithRepeatMode

dbWidthSpacingPatternId

```
dbCreateWidthSpacingPatternWithRepeatMode (
    dbCellViewId          cv,
    constString            name,
    dbWidthSpacingPatternSpec *patterns,
    unsigned int           numPatterns,
    dbDistance             offset,
    Boolean                isRepeatOffset,
    dbMaskColorType        startingColor,
    Boolean                isShiftColor
    dbWSPAllowedRepeatMode allowedMode,
    dbWSPRepeatMode        defaultMode
);
```

Description

Creates a width spacing pattern in the specified cellview keeping into consideration the enum values of allowed repeat pattern and default repeat pattern mode.

Arguments

<i>cv</i>	ID of the cellview in which the width spacing pattern is created.
<i>name</i>	Name for the width spacing pattern.
<i>patterns</i>	An array of pattern specifications.
<i>numPatterns</i>	Number of pattern specifications in the array of patterns.
<i>offset</i>	Distance of the first track from the period track.
<i>isRepeatOffset</i>	Flag to indicate whether the offset is applied when a pattern repeats.
<i>startingColor</i>	Color of the first track. If the color of the first track is not gray, all tracks are automatically colored such that the color pattern changes on each track. In other words, you cannot specify the color for a track other than the first one or leave a track uncolored.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

<i>isShiftColor</i>	Flag to indicate whether track colors change when a pattern repeats. This attribute is used only when all tracks are individually colored. If no color is specified for a track, it remains uncolored (that is, gray).
<i>startingColor</i>	Color of the first track. If the color of the first track is not gray, all tracks are automatically colored so that the color shifts on each track, that is there is no way to specify color for the tracks other than the first one or to leave some tracks uncolored.
<i>isShiftColor</i>	Flag to indicate whether track colors shift when a pattern repeats. This attribute is used only when all tracks are individually colored and no automatic color assignment is done. If no color is specified for a track, then it would be uncolored (that is gray).
<i>allowedMode</i>	Name of the allowed repeat pattern enum value to be set. Valid values: dbcWSPAllowedUnknown, dbcWSPAny, dbcWSPNone, dbcWSPSteppedOnly, dbcWSPFlippedOnly
<i>defaultMode</i>	Name of the default repeat pattern mode enum value to be set. Valid values: dbcWSPUnknown, dbcWSPNoRepeat, dbcWSPStepped, dbcWSPFlippedStartsWithOdd, dbcWSPFlippedStartsWithEven

Return Value

dbWidthSpacingPatternId ID of the created width spacing pattern.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbDeleteWidthSpacingPattern

Boolean

```
dbDeleteWidthSpacingPattern(  
    dbWidthSpacingPatternId wspId  
);
```

Description

Deletes the specified width spacing snap pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbFindWidthSpacingPattern

```
dbWidthSpacingPatternId  
dbFindWidthSpacingPattern(  
    dbCellViewId  cv,  
    constString   name  
);
```

Description

Searches for the width spacing pattern with the specified name in the specified cellview.

Arguments

<i>cv</i>	ID of the cellview in which the search is performed.
<i>name</i>	Name for the width spacing pattern for which the search is performed.

Return Value

<i>dbWidthSpacingPatternId</i>	ID of the width spacing pattern, if found. Returns <code>NULL</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbFreeWidthSpacingPatternSpecArray

```
void
dbFreeWidthSpacingPatternSpecArray(
    unsigned int          numPatternSpecs,
    dbWidthSpacingPatternSpec *patternSpecs
);
```

Description

Deallocates the memory allocated for an array of width spacing pattern specifications.

Note: In addition to deallocating the specified memory area, the function also frees the memory internally on the non-null `colors`, `wireTypes`, and `specs` members of each width spacing snap pattern specification in the array.

Arguments

<i>numPatternSpecs</i>	Size of the deallocated memory area, measured in blocks of <code>sizeof(dbWidthSpacingPatternSpec)</code> bytes.
<i>patternSpecs</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGenWidthSpacingPattern

Boolean

```
dbGenWidthSpacingPattern(  
    dbGenStateId          pState,  
    dbWidthSpacingPatternId *wspId  
);
```

Description

Retrieves the ID of the next width spacing pattern object from the generator with the given `dbGenStateId`. ID of the width spacing pattern object is returned in the `wspId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wspId</i>	ID of the next width spacing pattern object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the next width spacing pattern ID is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternAllowedRepeatMode

```
dbWSPAllowedRepeatMode  
dbGetWidthSpacingPatternAllowedRepeatMode (  
    dbWidthSpacingPatternId  wspId  
);
```

Description

Returns the allowed repeat mode for the specified width spacing pattern.

Arguments

<i>wspId</i>	ID of a width spacing pattern.

Return Value

<i>dbWSPAllowedRepeatMode</i>	Value of the allowed repeat mode enum value.

Example

```
dbGetWidthSpacingPatternAllowedRepeatMode (wspId)
```

Returns the allowed repeat mode enum value for the width spacing pattern *wspId*.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternDefaultRepeatMode

```
dbWSPRepeatMode  
dbGetWidthSpacingPatternDefaultRepeatMode (  
    dbWidthSpacingPatternId  wspId  
);
```

Description

Returns the default repeat mode of the specified width spacing pattern.

Arguments

<i>wspId</i>	ID of a width spacing pattern.

Return Value

<i>dbWSPRepeatMode</i>	Value of the default repeat mode enum value.

Example

```
dbGetWidthSpacingPatternDefaultRepeatMode (wspId)
```

Returns default repeat mode enum value of the width spacing pattern *wspId*.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternName

String

```
dbGetWidthSpacingPatternName (  
    dbWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the name for the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>String</i>	Name of the specified width spacing pattern object. Returns <code>NULL</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternOffset

```
dbDistance  
dbGetWidthSpacingPatternOffset(  
    dbWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the offset, which is the distance of the first track from the period track, for the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>dbDistance</i>	Offset value of the specified width spacing pattern object. Returns 0 in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternPatterns

```
void
dbGetWidthSpacingPatternPatterns (
    dbWidthSpacingPatternId    wspId,
    unsigned int                *numPatternSpecs,
    dbWidthSpacingPatternSpec   **patternSpecs
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of pattern specifications of the specified width spacing pattern object. Pointer to the allocated memory area is returned through the *patternSpecs* parameter and the size of this memory area is returned through the *numPatternSpecs* parameter.

Arguments

<i>wspId</i>	ID of the specified width spacing pattern object.
<i>numPatternSpecs</i>	Pointer to the memory area where the number of pattern specifications is placed.
<i>patternSpecs</i>	Pointer to the memory area where the array of pattern specifications is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternRepeatOffset

Boolean

```
dbGetWidthSpacingPatternRepeatOffset(  
    dbWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the repeat offset flag for the specified width spacing pattern object. The repeat offset flag is a Boolean value indicating whether the offset is applied when a pattern repeats.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>Boolean</i>	The repeat offset flag of the specified width spacing pattern object. Returns <code>FALSE</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternShiftColor

Boolean

```
dbGetWidthSpacingPatternShiftColor(  
    dbWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the shift color flag for the specified width spacing pattern object. The shift color flag is a Boolean value indicating whether the track colors shift when a pattern repeats.

Arguments

<i>wspId</i>	ID of the specified width spacing pattern object.

Return Value

<i>Boolean</i>	The shift color flag of the specified width spacing pattern object. Returns <code>FALSE</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternStartingColor

```
dbColorMaskType  
dbGetWidthSpacingPatternStartingColor(  
    dbWidthSpacingPatternId    wspId  
);
```

Description

Retrieves the color of the first track for the specified width spacing pattern object.

Arguments

<i>wspId</i>	ID of a width spacing pattern object.

Return Value

<i>dbColorMaskType</i>	Color of the first track.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbStartGenWidthSpacingPattern

```
dbGenStateId  
dbStartGenWidthSpacingPattern(  
    dbCellViewId  cvId  
);
```

Description

Initializes a generator for all width spacing pattern objects in the specified cellview. A generator state is returned, which is used with the [dbGenWidthSpacingPattern](#) function.

Arguments

<i>cvId</i>	ID of the cellview containing the width spacing pattern objects to be generated.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Width Spacing Pattern Group Functions

dbCreateWidthSpacingPatternGroup

```
dbWidthSpacingPatternGroupId  
dbCreateWidthSpacingPatternGroup(  
    dbCellViewId      cv,  
    constString        groupName,  
    const constString  *patternNames,  
    unsigned int        numPatternNames  
);
```

Description

Creates a group of width spacing patterns in the specified cellview.

Arguments

<i>cv</i>	ID of the cellview in which the width spacing pattern group is created.
<i>groupName</i>	Name of the width spacing pattern group.
<i>patternNames</i>	An array of pattern names to be included in the group.
<i>numPatternNames</i>	Number of elements in the array of pattern names.

Return Value

<i>dbWidthSpacingPatternGroupId</i>	ID of the created width spacing pattern group, if found.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbDeleteWidthSpacingPatternGroup

Boolean

```
dbDeleteWidthSpacingPatternGroup(  
    dbWidthSpacingPatternGroupId groupId  
);
```

Description

Deletes the specified width spacing pattern group object.

Arguments

<i>groupId</i>	ID of a width spacing pattern group object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbFindWidthSpacingPatternGroup

```
dbWidthSpacingPatternGroupId  
dbFindWidthSpacingPatternGroup(  
    dbCellViewId    cv,  
    constString     name  
);
```

Description

Searches for the width spacing pattern group with the specified name in the specified cellview.

Arguments

<i>cv</i>	ID of the cellview in which the search is performed.
<i>name</i>	Name of the width spacing pattern group for which the search is performed.

Return Value

dbWidthSpacingPatternId ID of the width spacing pattern group, if found.
Returns `NULL` in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGenWidthSpacingPatternGroup

Boolean

```
dbGenWidthSpacingPatternGroup(  
    dbGenStateId          pState,  
    dbWidthSpacingPatternGroupId *wspGroupId  
);
```

Description

Retrieves the ID of the next width spacing pattern group object from the generator with the given *dbGenStateId*. The width spacing pattern group object ID is returned in the *wspGroupId* argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wspGroupId</i>	ID of the next width spacing pattern group object produced by the generator. This value is undefined if FALSE is returned.

Return Value

<i>Boolean</i>	TRUE, if the ID of the next width spacing pattern group object is successfully generated. FALSE, if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternGroupName

String

```
dbGetWidthSpacingPatternGroupName (  
    dbWidthSpacingPatternGroupId  groupId  
);
```

Description

Retrieves the name for the specified width spacing pattern group object.

Arguments

<i>groupId</i>	ID of the specified width spacing pattern group object.

Return Value

<i>String</i>	Name of the specified width spacing pattern group object.
	Returns NULL in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingPatternGroupPatternNames

```
void
dbGetWidthSpacingPatternGroupPatternNames (
    dbWidthSpacingPatternGroupId    groupId,
    unsigned int                    *numPatternNames,
    String                          **patternNames
);
```

Description

Allocates a suitable amount of heap memory and fills it with the array of pattern names of the specified width spacing pattern group. Pointer to the allocated memory area is returned through the *patternsName* parameter and the size of this memory area is returned through the *numPatternNames* parameter.

Arguments

<i>groupId</i>	ID of the specified width spacing pattern group object.
<i>numPatternNames</i>	Pointer to the memory area where the number of pattern names is placed.
<i>patternsName</i>	Pointer to the memory area where the array of pattern names is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbStartGenWidthSpacingPatternGroup

```
dbGenStateId  
dbStartGenWidthSpacingPatternGroup(  
    dbCellViewId    cvId  
);
```

Description

Initializes a generator for all width spacing pattern group objects in the specified cellview. A generator state is returned, which is used with the [dbGenWidthSpacingPatternGroup](#) function.

Arguments

<i>cvId</i>	ID of the cellview containing the width spacing pattern group objects to be generated.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Pattern Region Functions

dbGetPatternRegionCommittedPatterns

```
void  
dbGetPatternRegionCommittedPatterns(  
    dbShapeId      shapeId,  
    String         **patternNames,  
    unsigned int    *numPatterns  
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of committed pattern names of the specified pattern region object. Pointer to the allocated memory area is returned through the *patternNames* parameter and the size of this memory area is returned through the *numPatterns* parameter.

Arguments

<i>shapeId</i>	ID of the shape for which committed patterns are retrieved.
<i>patternNames</i>	Pointer to the memory area where the array of committed patterns is placed.
<i>numPatterns</i>	Pointer to the memory area where the number of committed patterns is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetPatternRegionActivePattern

String

```
dbGetPatternRegionActivePattern(  
    dbShapeId      shapeId  
);
```

Description

Retrieves the name of the active pattern currently set for the specified shape.

Arguments

<i>shapeId</i>	ID of the shape for which the active pattern is to be retrieved.

Return Value

<i>String</i>	Name of the active pattern.

dbGetPatternRegionAllowedPatternGroups

```
void  
dbGetPatternRegionAllowedPatternGroups (  
    dbShapeId      shapeId,  
    String          **groupNames,  
    unsigned int    *numGroups  
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of allowed pattern group names of the specified pattern region object. Pointer to the allocated memory area is returned through the *groupNames* parameter and the size of this memory area is returned through the *numGroups* parameter.

Arguments

<i>shapeId</i>	ID of the shape.
<i>groupNames</i>	Pointer to the memory area where the array of allowed pattern groups is placed.
<i>numGroups</i>	Pointer to the memory area where the number of allowed pattern groups is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetPatternRegionAllowedPatterns

```
void
dbGetPatternRegionAllowedPatterns (
    dbShapeId      shapeId,
    String          **patternNames,
    unsigned int    *numPatterns
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of allowed pattern names of the specified pattern region object. Pointer to the allocated memory area is returned through the *patternNames* parameter and the size of this memory area is returned through the *numPatterns* parameter.

Arguments

<i>shapeId</i>	ID of the shape.
<i>patternNames</i>	Pointer to the memory area where the array of allowed patterns is placed.
<i>numPatterns</i>	Pointer to the memory area where the number of allowed patterns is placed.

Return Value

No value returned.

dbGetPatternRegionCreateType

String

```
dbGetPatternRegionCreateType(  
    dbShapeId    shapeId  
);
```

Description

Retrieves the create type for the specified shape. The return value is either "systemCreated" or "userCreated".

Arguments

<i>shapeId</i>	ID of the shape.

Return Value

<i>String</i>	Create type of the pattern region.

dbGetPatternRegionDefaultRepeatMode

```
dbWSPRepeatMode  
dbGetPatternRegionDefaultRepeatMode (  
    dbShapeId    shapeId  
);
```

Description

Returns the default repeat mode for the specified pattern region.

Arguments

<i>shapeId</i>	ID of the pattern region for which the repeat mode is set.

Return Value

<i>dbWSPRepeatMode</i>	The default repeat mode enum value for the specified pattern region.

Example

```
dbGetPatternRegionDefaultRepeatMode (shapeId)
```

Returns the default repeat mode enum value for the pattern region *shapeId*.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetPatternRegionRepeatMode

dbWSPRepeatMode

```
dbGetPatternRegionRepeatMode (
    dbShapeId      shapeId,
    constString    wspName,
    );
```

Description

Returns the repeat mode for the specified width spacing pattern in the given pattern region.

Arguments

<i>shapeId</i>	ID of the pattern region for which the repeat mode is set.
<i>wspName</i>	Name of a width spacing pattern.

Return Value

<i>dbWSPRepeatMode</i>	The repeat mode enum value for the specified pattern region.

Example

```
dbGetPatternRegionRepeatMode (shapeId, "wspName")
```

Returns the repeat mode enum value for the width spacing pattern *wspName* in the pattern region *shapeId*.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetPatternRegionShiftColor

Boolean

```
dbGetPatternRegionShiftColor(  
    dbShapeId    shapeId  
);
```

Description

Retrieves the shift color flag for the specified shape.

Arguments

<i>shapeId</i>	ID of the shape.

Return Values

<i>Boolean</i>	Shift color flag of the shape. Returns <code>FALSE</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbHasPatternRegionActivePattern

Boolean

```
dbHasPatternRegionActivePattern(  
    dbShapeId    shapeId  
);
```

Description

Checks if an active pattern is set for the specified shape.

Arguments

<i>shapeId</i>	ID of the shape.

Return Values

<i>Boolean</i>	TRUE, if an active pattern is set for the shape. FALSE, if the shape does not have an active pattern set or in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetPatternRegionActivePattern

Boolean

```
dbSetPatternRegionActivePattern(  
    dbShapeId      shapeId,  
    constString    patternName)  
);
```

Description

Sets the active pattern for the specified shape.

Arguments

<i>shapeId</i>	ID of the shape.
<i>patternName</i>	Name of the pattern to be set as active.

Return Values

<i>Boolean</i>	TRUE, if active pattern is set. FALSE, in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetPatternRegionAllowedPatternGroups

Boolean

```
dbSetPatternRegionAllowedPatternGroups (
    dbShapeId      shapeId,
    String          **groupNames,
    unsigned int    *numGroups
);
```

Description

Sets allowed pattern groups for the specified shape.

Arguments

<i>shapeId</i>	ID of the shape.
<i>groupNames</i>	Array of allowed pattern groups to be set.
<i>numGroups</i>	Number of elements in the array of group names.

Return Values

<i>Boolean</i>	TRUE, if the allowed pattern groups are set. FALSE, in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetPatternRegionAllowedPatterns

Boolean

```
dbSetPatternRegionAllowedPatterns (
    dbShapeId      shapeId,
    consString     **patternNames,
    unsigned int   *numPatterns
);
```

Description

Sets the allowed patterns for the specified shape.

Arguments

<i>shapeId</i>	ID of the shape.
<i>patternNames</i>	Array of allowed patterns to be set.
<i>numPatterns</i>	Number of elements in the array of patterns.

Return Values

<i>Boolean</i>	TRUE, if the allowed patterns are set. FALSE, in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetPatternRegionDefaultRepeatMode

Boolean

```
SetPatternRegionDefaultRepeatMode (
    dbShapeId      shapeId,
    dbWSPRepeatMode repeatMode
);
```

Description

Sets the default repeat mode on the specified pattern region.

Arguments

<i>shapeId</i>	ID of the pattern region for which the repeat mode is to be set.
<i>repeatMode</i>	Name of the repeat mode enum value to be set. Valid values: dbcWSPNoRepeat, dbcWSPStepped, dbcWSPFlippedStartsWithOdd, dbcWSPFlippedStartsWithEven

Value Returned

<i>Boolean</i>	TRUE, if the sepecified repeat mode is successfully set on the pattern region. FALSE, in case of failure.

Example

```
dbSetPatternRegionDefaultRepeatMode(shapeID, dbcWSPStepped)
```

Sets the default repeat mode as dbcWSPStepped for the pattern region shapeID.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetPatternRegionRepeatMode

Boolean

```
dbSetPatternRegionRepeatMode (
    dbShapeId      shapeId,
    constString    wspName,
    dbWSPRepeatMode repeatMode
);
```

Description

Sets the specified repeat mode on the pattern region for the specified width spacing pattern.

Arguments

<i>shapeId</i>	ID of the pattern region on which the repeat mode is to be set.
<i>wspName</i>	Name of a width spacing pattern.
<i>repeatMode</i>	Name of the repeat mode enum value to be set. Valid values: dbcWSPNoRepeat, dbcWSPStepped, dbcWSPFlippedStartsWithOdd, dbcWSPFlippedStartsWithEven

Return Value

<i>Boolean</i>	TRUE, if the repeat mode is successfully set on the pattern region for the specified width spacing pattern. FALSE, in case of failure.

Example

```
dbSetPatternRegionRepeatMode(shapeId, "wspName", dbcWSPStepped)
```

In this example, repeat mode `dbcWSPStepped` is successfully set on the pattern region `shapeId` for the width spacing pattern `wspName`.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbShapeHasPatternRegionInfo

Boolean

```
dbShapeHasPatternRegionInfo(  
    dbShapeId    shapeId  
);
```

Description

Returns a Boolean value indicating whether the specified shape contains pattern region information.

Arguments

<i>shapeId</i>	ID of the shape.

Return Values

<i>Boolean</i>	TRUE, if the shape contains pattern region information. FALSE, if the shape does not contain pattern region information or in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetPatternRegionName

```
constString  
dbGetPatternRegionName (  
    dbShapeId    shapeId  
);
```

Description

Returns the name of the pattern region set for the specified shape.

Arguments

<i>shapeId</i>	Database ID of a shape.

Value Returned

<i>constString</i>	Name of the pattern region.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbSetPatternRegionName

Boolean

```
dbSetPatternRegionName(  
    dbShapeId  shapeId,  
    constString name.  
);
```

Description

Sets the specified pattern region for the specified shape.

Arguments

<i>shapeId</i>	Database ID of a shape.
<i>name</i>	Name of a pattern region.

Value Returned

Boolean	TRUE if the specified pattern region is set. FALSE if the specified region is not set or an error occurs.

dbGetPatternRegionCommittedPatternShiftColor

Boolean

```
dbGetPatternRegionCommittedPatternShiftColor(  
    dbShapeId  shapeId,  
    constString patternName  
);
```

Description

Returns a shiftColor flag of the committed pattern with the specified name.

Arguments

<i>shapeId</i>	ID of the shape for which the committed patterns are retrieved.
<i>patternName</i>	Name of the committed pattern.

Return Value

Boolean	TRUE if the shiftColor flag of the committed pattern is retrieved. FALSE if the shiftColor flag of the committed pattern is not retrieved or an error occurs.

Width Spacing Snap Pattern Def Functions

dbAllocSnappingLayerArray

```
dbSnappingLayer*  
dbAllocSnappingLayerArray(  
    unsigned int    numLayers  
);
```

Description

Allocates memory for the array of snapping layers and initializes the memory with zeroes.

Arguments

<i>numLayers</i>	Number of snapping layers to fit in the allocated memory.

Return Value

<i>dbSnappingLayer*</i>	Pointer to the allocated memory area.

dbFreeSnappingLayerArray

```
void  
dbFreeSnappingLayerArray(  
    unsigned int    numLayers,  
    dbSnappingLayer *layers  
);
```

Description

Deallocates the memory allocated for an array of snapping layers.

Arguments

<i>numLayers</i>	Size of the deallocated memory area, measured in blocks of <code>sizeof(dbSnappingLayer)</code> bytes.
<i>layers</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbCreateWidthSpacingSnapPatternDef

```
dbWidthSpacingSnapPatternDefId
dbCreateWidthSpacingSnapPatternDef (
    dbCellViewId      dbId,
    constString        name,
    dbLayer            layerNum,
    dbPurpose          purposeNum,
    dbDistance         period,
    Boolean            isHorizontal,
    dbDistance         offset,
    dbSnappingLayer    *snappingLayers,
    unsigned int        numSnappingLayers,
    String             *patternNames,
    unsigned int        numPatterns,
    String             *patternGroupNames,
    unsigned int        numPatternGroups,
    String             defaultActive
);
```

Description

Creates a width spacing snap pattern definition object in the specified database.

Arguments

<i>dbId</i>	ID of the database in which the width spacing snap pattern definition object is created.
<i>name</i>	Name for the width spacing snap pattern definition object.
<i>layerNum</i>	Layer number for the width spacing snap pattern definition object.
<i>purposeNum</i>	Purpose number for the width spacing snap pattern definition object.
<i>period</i>	Spacing between coarse-grain period tracks (heart-beat period).
<i>isHorizontal</i>	Pattern direction: TRUE for horizontal and FALSE for vertical.
<i>offset</i>	Distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

<i>snappingLayers</i>	Array of snapping layers for the width spacing snap pattern definitions.
<i>numSnappingLayers</i>	Number of elements in the array of snapping layers..
<i>patternNames</i>	Array of allowed width spacing pattern names.
<i>numPatterns</i>	Number of elements in the array of patterns names.
<i>patternGroupNames</i>	Array of allowed width spacing pattern group names.
<i>numPatternGroups</i>	Number of elements in the array of pattern group names.
<i>defaultActive</i>	Pattern to be used in areas where no region has been drawn.

Return Value

bWidthSpacingSnapPatternDefId

<i>d</i>	ID of the created width spacing snap pattern definition object.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbDeleteWidthSpacingSnapPatternDef

Boolean

```
dbDeleteWidthSpacingSnapPatternDef(  
    dbWidthSpacingSnapPatternDefId spDefId  
);
```

Description

Deletes the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbFindWidthSpacingSnapPatternDefByName

```
dbWidthSpacingSnapPatternDefId  
dbFindWidthSpacingSnapPatternDefByName (  
    dbCellViewId dbId,  
    constString name  
);
```

Description

Searches for the width spacing snap pattern definition object with the specified name in the specified database.

Arguments

<i>dbId</i>	ID of the database in which the search is performed.
<i>name</i>	Name of the width spacing pattern for which the search is performed.

Return Value

<i>dbWidthSpacingSnapPatternDefId</i>	ID of the width spacing snap pattern definition object, if found. Returns <code>NULL</code> in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGenWidthSpacingSnapPatternDef

Boolean

```
dbGenWidthSpacingSnapPatternDef(  
    dbGenStateId          pState,  
    dbWidthSpacingSnapPatternDefId *wsspDefId  
);
```

Description

Retrieves the ID of the next width spacing snap pattern definition object from the generator with the given `dbGenStateId`. ID of the width spacing snap pattern definition object is returned in the `wsspDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wsspDefId</i>	ID of the next width spacing snap pattern definition object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the ID of the next width spacing snap pattern definition object is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGenWidthSpacingSnapPatternDefForLPP

Boolean

```
dbGenWidthSpacingSnapPatternDefForLPP(  
    dbGenStateId          pState,  
    dbWidthSpacingSnapPatternDefId  *wsspDefId  
);
```

Description

Retrieves the ID of the next width spacing snap pattern definition object with the given `dbGenStateId`. ID of the width spacing snap pattern definition object is returned in the `wsspDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wsspDefId</i>	ID of the next width spacing snap pattern definition object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the ID of the next width spacing snap pattern definition object is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGenWidthSpacingSnapPatternDefForSnappingLPP

Boolean

```
dbGenWidthSpacingSnapPatternDefForSnappingLPP(  
    dbGenStateId          pState,  
    dbWidthSpacingSnapPatternDefId  *wsspDefId  
);
```

Description

Retrieves the ID of the next width spacing snap pattern definition object with the given `dbGenStateId`. ID of the width spacing snap pattern definition object is returned in the `wsspDefId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>wsspDefId</i>	ID of the next width spacing snap pattern definition object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the ID of the next width spacing snap pattern definition object is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefDefaultActiveName

String

```
dbGetWidthSpacingSnapPatternDefDefaultActiveName (  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the name of the default active pattern for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>String</i>	Default active pattern name of the specified width spacing snap pattern definition object. Returns NULL in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefLayer

```
dbLayer  
dbGetWidthSpacingSnapPatternDefLayer(  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the layer number for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>dbLayer</i>	The layer number for the specified width spacing snap pattern definition object. Returns <code>dbcNullLayer</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefName

String

```
dbGetWidthSpacingSnapPatternDefName (  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the name for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>String</i>	Name of the specified width spacing snap pattern definition object. Returns NULL in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefOffset

```
dbDistance  
dbGetWidthSpacingSnapPatternDefOffset(  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the offset for the specified width spacing snap pattern definition object. This is the distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>dbDistance</i>	The offset value of the specified width spacing snap pattern definition object.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefPatternGroupNames

```
void
dbGetWidthSpacingSnapPatternDefPatternGroupNames (
    dbWidthSpacingSnapPatternDefId      spDefId,
    unsigned int                        *numPatternGroups,
    String                             **patternGroupNames
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of pattern group names of the specified width spacing snap pattern definition object. Pointer to the allocated memory area is returned through the *patternGroupNames* parameter and the size of this memory area is returned through the *numPatternGroups* parameter.

Arguments

<i>spDefId</i>	ID of the specified width spacing snap pattern definition object.
<i>numPatternGroups</i>	Pointer to the memory area where the number of pattern group names is placed.
<i>patternGroupNames</i>	Pointer to the memory area where the array of pattern group names is placed.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefPatternNames

```
void
dbGetWidthSpacingSnapPatternDefPatternNames (
    dbWidthSpacingSnapPatternDefId      spDefId,
    unsigned int                        *numPatterns,
    String                              **patternNames
);
```

Description

Allocates a suitable amount of heap memory, and fills it with the array of pattern names of the specified width spacing snap pattern definition object. Pointer to the allocated memory area is returned through the *patternNames* parameter and the size of this memory area is returned through the *numPatterns* parameter.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.
<i>numPatterns</i>	Pointer to the memory area where the number of pattern names is placed.
<i>patternNames</i>	Pointer to the memory area where the array of pattern names is placed.

Return Value

No return value.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefPeriod

```
dbDistance  
dbGetWidthSpacingSnapPatternDefPeriod(  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the spacing between coarse-grain period tracks for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of the specified width spacing snap pattern definition object.

Return Value

<i>dbDistance</i>	Spacing between coarse-grain period tracks.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefPurpose

```
dbPurpose  
dbGetWidthSpacingSnapPatternDefPurpose (  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Retrieves the purpose for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of the specified width spacing snap pattern definition object.

Return Value

<i>dbPurpose</i>	The purpose number for the specified width spacing snap pattern definition object. Returns <code>dbcUnknownPurpose</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetWidthSpacingSnapPatternDefSnappingLayers

```
void
dbGetWidthSpacingSnapPatternDefSnappingLayers (
    dbWidthSpacingSnapPatternDefId  spDefId,
    unsigned int                    *numSnappingLayers,
    dbSnappingLayer                 **snappingLayers
);
```

Description

Retrieves an array of snapping layers for the specified width spacing snap pattern definition object.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.
<i>numSnappingLayers</i>	Pointer to the memory area where the number of snapping layers is placed.
<i>snappingLayers</i>	Pointer to the memory area where the array of snapping layers is placed.

Return Value

No value returned.

dbIsWidthSpacingSnapPatternDefHorizontal

Boolean

```
dbIsWidthSpacingSnapPatternDefHorizontal(  
    dbWidthSpacingSnapPatternDefId    spDefId  
);
```

Description

Verifies whether the direction of the specified width spacing snap pattern definition object is horizontal.

Arguments

<i>spDefId</i>	ID of a width spacing snap pattern definition object.

Return Value

<i>Boolean</i>	TRUE, if the object direction is horizontal. FALSE, if the object direction is vertical.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbStartGenWidthSpacingSnapPatternDef

```
dbGenStateId  
dbStartGenWidthSpacingSnapPatternDef(  
    dbCellViewId    dbId  
);
```

Description

Initializes a generator for all width spacing snap pattern definition objects in the specified cellview. A generator state is returned, which is used with the dbGenWidthSpacingSnapPatternDef function.

Arguments

<i>dbId</i>	ID of the cellview containing the width spacing snap pattern definition objects to be generated.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbStartGenWidthSpacingSnapPatternDefForSnappingLPP

```
dbGenStateId  
dbStartGenWidthSpacingSnapPatternDefForSnappingLPP(  
    dbCellViewId dbId,  
    dbLayer      layer,  
    dbPurpose    purpose  
);
```

Description

Initializes a generator for all width spacing snap pattern definition objects in the specified cellview that have the specified snapping LPP. A generator state is returned, which is used with the dbGenWidthSpacingSnapPatternDefForSnappingLPP function.

Arguments

<i>dbId</i>	ID of the cellview containing the width spacing snap pattern definition objects to be generated.
<i>layer</i>	Layer ID in a layer-purpose pair.
<i>purpose</i>	Purpose ID in a layer-purpose pair.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Related Snap Pattern Functions

dbAllocRelatedSnapPatternsSpecArray

```
dbRelatedSnapPatternsSpec*  
dbAllocRelatedSnapPatternsSpecArray(  
    unsigned int    numSpecs  
);
```

Description

Allocates memory for the array of related snap patterns specifications and initializes the memory with zeroes.

Arguments

<i>numSpecs</i>	The number of related snap patterns specifications to fit in the allocated memory.

Return Value

<i>dbRelatedSnapPatternsSpec*</i>	Pointer to the allocated memory area.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbCreateRelatedSnapPatterns

dbRelatedSnapPatternsId

```
dbCreateRelatedSnapPatterns (
    dbCellViewId          cv,
    constString            name,
    const dbRelatedSnapPatternsSpec *relatedSnapPatterns,
    unsigned int           numRelatedSnapPatterns
);
```

Description

Creates the specified related snap patterns object in the specified cellview.

Arguments

<i>cv</i>	ID of the cellview in which the related snap patterns object is created.
<i>name</i>	Name of a related snap patterns object.
<i>relatedSnapPatterns</i>	Array of related snap pattern names.
<i>numRelatedSnapPatterns</i>	Number of elements in the array of related snap patterns.

Return Value

<i>dbRelatedSnapPatternsId</i>	ID of the created related snap patterns object.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbDeleteRelatedSnapPatterns

Boolean

```
dbDeleteRelatedSnapPatterns (
    dbRelatedSnapPatternsId    rspId
);
```

Description

Deletes the specified related snap patterns object.

Arguments

<i>rspId</i>	ID of the related snap patterns object.

Return Value

<i>Boolean</i>	TRUE, if the object was successfully deleted. FALSE, if the object was not deleted.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbFindRelatedSnapPatterns

```
dbRelatedSnapPatternsId  
dbFindRelatedSnapPatterns(  
    dbCellViewId  cv,  
    constString   name  
);
```

Description

Searches for the related snap patterns object with the specified name in the specified cellview.

Arguments

<i>cv</i>	ID of the cellview in which the search is performed.
<i>name</i>	Name of the related snap patterns object for which the search is performed.

Return Value

dbRelatedSnapPatternsId ID of the related snap patterns object, if found.
Returns NULL in case of failure.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbFreeRelatedSnapPatternsSpecArray

```
void
dbFreeRelatedSnapPatternsSpecArray(
    unsigned int          numSpecs,
    dbRelatedSnapPatternsSpec *patternSpecs
);
```

Description

Frees the memory allocated for an array of related snap patterns specifications.

Arguments

<i>numSpecs</i>	Size of the deallocated memory area, measured in blocks of <i>sizeof(dbRelatedSnapPatternsSpec)</i> bytes.
<i>patternSpecs</i>	Pointer to the memory area to deallocate.

Return Value

No value returned.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGenRelatedSnapPatterns

Boolean

```
dbGenRelatedSnapPatterns (
    dbGenStateId          pState,
    dbRelatedSnapPatternsId *rSPId
);
```

Description

Retrieves the ID of the next related snap patterns object from the generator with the given `dbGenStateId`. ID of the related snap patterns object is returned in the *rSPId* argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>rSPId</i>	ID of the next related snap patterns object produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if ID of the next related snap patterns object is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetRelatedSnapPatternsName

String

```
dbGetRelatedSnapPatternsName (  
    dbRelatedSnapPatternsId  relatedSnapPatternsId  
);
```

Description

Retrieves the name for the specified related snap patterns object.

Arguments

<i>relatedSnapPatternsId</i>	ID of the related snap patterns object.

Return Value

<i>String</i>	Name of the specified related snap patterns object. Returns <code>NULL</code> in case of error.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

dbGetRelatedSnapPatternsSpec

```
void
dbGetRelatedSnapPatternsSpec (
    dbRelatedSnapPatternsId    relatedSnapPatternsId,
    unsigned int                *numSpecs,
    dbRelatedSnapPatternsSpec  **patternSpecs
);
```

Description

Retrieves the array of related snap pattern specifications for the specified related snap patterns object.

Arguments

<i>relatedSnapPatternsId</i>	ID of a related snap patterns object.
<i>numSpecs</i>	Pointer to the memory area where the number of related snap pattern specifications is placed.
<i>patternSpecs</i>	Pointer to the memory area where the array of related snap pattern specifications is placed.

Return Value

No value returned.

dbStartGenRelatedSnapPatterns

```
dbGenStateId  
dbStartGenRelatedSnapPatterns(  
    dbCellViewId    cvId  
);
```

Description

Initializes a generator for all related snap patterns objects in the specified cellview. A generator state is returned, which is used with the [dbDeleteRelatedSnapPatterns](#) function.

Arguments

<i>cvId</i>	ID of the cellview containing the related snap patterns objects to be generated.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

Example- Using WSP Database Functions

```
// Create a scratch cellview.
ddId          libId = ddCreateLib("design", getTestDir() + "/design");
dbCellViewId  cellview = dbOpenCellViewByType("design", "top", "layout",
                                              "maskLayout", "w",
                                              dbcNullCellViewId);

// *** WIDTH SPACING PATTERN API EXAMPLES ***

// Create the test data.
const unsigned int      num_wsps = 3;
constString             *wspNames = calloc(num_wsps,
                                           constString);

wspNames[0] = strdup("1X");
wspNames[1] = strdup("2X");
wspNames[2] = strdup("1X2X");

const unsigned int      num_colors = 2;
dbMaskColorType         *colors = calloc(num_colors,
                                          dbMaskColorType);

colors[0] = dbcGrayColor;
colors[1] = dbcMask3Color;

const unsigned int      num_wireTypes = 2;
constString             *wireTypes = calloc(num_wireTypes,
                                           constString);

wireTypes[0] = strdup("wire0");
wireTypes[1] = strdup("wire1");

const unsigned int      num_baseSpecs = 2;
dbWidthSpacingSpec      *baseSpecs = calloc(num_baseSpecs,
                                           dbWidthSpacingSpec);

baseSpecs[0] = { 20, 20, dbcGrayColor, (String) wireTypes[0] };
baseSpecs[1] = { 10, 10, dbcGrayColor, (String) wireTypes[1] };
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
const unsigned int      num_wspSpecs = 2;
dbWidthSpacingPatternSpec *wspSpecs
= dbAllocWidthSpacingPatternSpecArray(num_wspSpecs);

// Note: dbFreeWidthSpacingPatternSpecArray() deallocates all members of the
// array and assumes that all members were allocated on the heap. This means
// that dbWidthSpacingPatternSpec members must be allocated with a heap
// allocation function such as calloc().

wspSpecs[0] = {3, NULL, 0, NULL, 0, baseSpecs, num_baseSpecs };
wspSpecs[1] = {3, colors, num_colors, (String *) wireTypes, num_wireTypes,
               NULL, 0 };

const unsigned int      num_wspSpecs_invalid = 2;
dbWidthSpacingPatternSpec *wspSpecs_invalid
= dbAllocWidthSpacingPatternSpecArray(num_wspSpecs_invalid);

dbMaskColorType         *colors2 = calloc(num_colors,
                                           dbMaskColorType);
memcpy(colors2, colors, num_colors * sizeof(dbMaskColorType));

constString             *wireTypes2 = calloc(num_wireTypes,
                                              constString);
memcpy(wireTypes2, wireTypes, num_wireTypes * sizeof(constString));

dbWidthSpacingSpec      *baseSpecs2 = calloc(num_baseSpecs,
                                              dbWidthSpacingSpec);
dbWidthSpacingSpec      *baseSpecs3 = calloc(num_baseSpecs,
                                              dbWidthSpacingSpec);

memcpy(baseSpecs2, baseSpecs, num_baseSpecs * sizeof(dbWidthSpacingSpec));
memcpy(baseSpecs3, baseSpecs, num_baseSpecs * sizeof(dbWidthSpacingSpec));

// This spec is invalid because it specifies the list of repeat colors
// both in the widthSpacingSpec and in the widthSpacingPattern spec.

wspSpecs_invalid[0] = {3, colors2, num_colors, NULL, 0,
                      baseSpecs2, num_baseSpecs };
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
// This spec is invalid because it specifies the list of wire types
// both in the widthSpacingSpec and in the widthSpacingPattern spec.

wspSpecs_invalid[1] = {3, NULL, 0, (String *) wireTypes2, num_wireTypes,
                      baseSpecs3, num_baseSpecs };

// These commands will fail due to the invalid specs.
dbCreateWidthSpacingPattern(cellview, "bad_wsp1", wspSpecs_invalid, 1,
                           0, FALSE, dbcGrayColor, FALSE);

dbCreateWidthSpacingPattern(cellview, "bad_wsp2", wspSpecs_invalid + 1, 1,
                           0, FALSE, dbcGrayColor, FALSE);

// This command will fail because we cannot define both the startingColor
// parameter and the list of repeat colors in the specification.
dbCreateWidthSpacingPattern(cellview, "bad_wsp3", wspSpecs, 2,
                           0, FALSE, dbcMask3Color, FALSE);

// Generate valid widthSpacingPatterns.
dbWidthSpacingPatternId wsps[num_wsps];

for (int i = 0; i < num_wsps; i++) {
    wsps[i] = dbCreateWidthSpacingPattern(cellview, wspNames[i], wspSpecs,
                                         num_wspSpecs, 0, FALSE,
                                         dbcGrayColor, FALSE);

    assert(wsps[i]);
    dbWidthSpacingPatternId wsp_repeat;

    wsp_repeat = dbCreateWidthSpacingPatternWithRepeatMode(cellview, "wsp_repeat",
                                                           wspSpecs, num_wspSpecs,
                                                           0, FALSE, dbcGrayColor,
                                                           FALSE, dbcWSPSteppedOnly,
                                                           dbcWSPStepped); assert(wsp_repeat);
}

// Examples for the getter functions.
cout << "Sample WSP dump: " << endl;
cout << "  name: " << dbGetWidthSpacingPatternName(wsps[0]) << endl;
cout << "  offset: " << dbGetWidthSpacingPatternOffset(wsps[0]) << endl;
cout << "  repeat offset: " << dbGetWidthSpacingPatternRepeatOffset(wsps[0])
<< endl;
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
cout << "  shift color: " << (bool) dbGetWidthSpacingPatternShiftColor(wsps[0])
<< endl;
cout << "  starting color: "
<< dbMaskColorTypeToString(dbGetWidthSpacingPatternStartingColor(wsps[0])) <<
endl;
cout << endl << endl;

unsigned int          tmp_num = 0;
dbWidthSpacingPatternSpec *tmp_specs = NULL;

dbGetWidthSpacingPatternPatterns(wsps[0], &tmp_num, &tmp_specs);
assert(tmp_num == num_wspSpecs);
dbFreeWidthSpacingPatternSpecArray(tmp_num, tmp_specs);

// Examples for the generator functions.
dbGenStateId          gen_state = dbStartGenWidthSpacingPattern(cellview);
dbWidthSpacingPatternId tmp_wsp;

cout << "Pattern names returned by the generator: ";

while (dbGenWidthSpacingPattern(gen_state, &tmp_wsp))
    cout << " " << dbGetWidthSpacingPatternName(tmp_wsp);

cout << endl << endl;
dbStopGen(gen_state);

// Examples for the find and delete functions.
assert(dbFindWidthSpacingPattern(cellview, "tmp_wsp")
    == dbcNullWidthSpacingPatternId);
tmp_wsp = dbCreateWidthSpacingPattern(cellview, "tmp_wsp", wspSpecs,
    num_wspSpecs, 0, FALSE,
    dbcGrayColor, FALSE);
assert(dbFindWidthSpacingPattern(cellview, "tmp_wsp"));
assert(dbDeleteWidthSpacingPattern(tmp_wsp));
assert(dbFindWidthSpacingPattern(cellview, "tmp_wsp")
    == dbcNullWidthSpacingPatternId);

// *** WIDTH SPACING PATTERN GROUP API EXAMPLES ***

constString          wspgNames[] = {"wspg1"};
const unsigned int    num_wspgs = ARRAY_SIZE(wspgNames);
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
// This command will fail because a nonexisting widthSpacingPattern name
// is specified.
constString          bad_wsp = "bad_wsp";

dbCreateWidthSpacingPatternGroup(cellview, NULL, (String *) &bad_wsp, 1);

// Generate valid widthSpacingPatternGroups.
dbWidthSpacingPatternGroupId    wspgs[num_wspgs];

for (int i = 0; i < num_wspgs; i++) {
    wspgs[i] = dbCreateWidthSpacingPatternGroup(cellview, wspgNames[i],
                                                (String *) &wspNames[0],
                                                num_wspgs);

    assert(wspgs[i]);
}

// Examples for the getter functions.
cout << "Sample WSP group dump: " << endl;
cout << "  name: " << dbGetWidthSpacingPatternGroupName(wspgs[0]) << endl;
cout << "  patterns:";

constString*  tmp_names = NULL;

dbGetWidthSpacingPatternGroupPatternNames(wspgs[0], &tmp_num, &tmp_names);
assert(tmp_num == num_wspgs);

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl << endl;

free(tmp_names);

// Examples for the generator functions.
gen_state = dbStartGenWidthSpacingPatternGroup(cellview);
dbWidthSpacingPatternGroupId tmp_wspg;

cout << "Pattern group names returned by the generator: ";
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
while (dbGenWidthSpacingPatternGroup(gen_state, &tmp_wspg))
    cout << " " << dbGetWidthSpacingPatternGroupName(tmp_wspg);

cout << endl << endl;
dbStopGen(gen_state);

// Examples for the find and delete functions.
assert(dbFindWidthSpacingPatternGroup(cellview, "tmp_wspg")
    == dbcNullWidthSpacingPatternGroupId);
tmp_wspg = dbCreateWidthSpacingPatternGroup(cellview, "tmp_wspg",
    (String *) &wspNames[0], 1);
assert(dbFindWidthSpacingPatternGroup(cellview, "tmp_wspg"));
assert(dbDeleteWidthSpacingPatternGroup(tmp_wspg));
assert(dbFindWidthSpacingPatternGroup(cellview, "tmp_wspg")
    == dbcNullWidthSpacingPatternGroupId);

// *** PATTERN REGION API EXAMPLES ***
dbBBox bBox;

dbFillBBox(&bBox, 0, 0, 200, 100);

dbRectId rect = dbCreateRect(cellview, 10, dbcDrawingPurpose, &bBox);

// Since "rect" is a normal figure, this command will return FALSE.
assert(dbShapeHasPatternRegionInfo(rect) == FALSE);

// Examples for the dbSetPatternRegion...() functions.
// Using any dbSetPatternRegion... command on a dbShapeId initializes pattern
// region data, so dbShapeHasPatternRegionInfo() will be returning TRUE.
dbSetPatternRegionActivePattern(rect, wspNames[0]);
assert(dbShapeHasPatternRegionInfo(rect) == TRUE);
assert(dbHasPatternRegionActivePattern(rect) == TRUE);
dbSetPatternRegionAllowedPatterns(rect, wspNames, num_wsps);
dbSetPatternRegionAllowedPatternGroups(rect, wspgNames, num_wspgs);

// Examples for the dbGetPatternRegion...() functions.
cout << "Sample patternRegion dump: " << endl;
cout << " create type: "
<< dbGetPatternRegionCreateType(rect) << endl;
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
cout << "  shift color: "
<< (bool) dbGetPatternRegionShiftColor(rect) << endl;

dbGetPatternRegionAllowedPatterns(rect, &tmp_names, &tmp_num);
assert(tmp_num == num_wsps);
cout << "  patterns:";

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);
dbGetPatternRegionAllowedPatternGroups(rect, &tmp_names, &tmp_num);
assert(tmp_num == num_wspgs);
cout << "  pattern groups:";

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);
dbGetPatternRegionCommittedPatterns(rect, &tmp_names, &tmp_num);
assert(tmp_num == num_wsps);
cout << "  committed patterns:";

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);

cout << "  active pattern: "
<< dbGetPatternRegionActivePattern(rect) << endl;

// *** WIDTH SPACING SNAP PATTERN DEF API EXAMPLES ***

// Create the test data.
```


Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
// At this point, we need the technology database.

techFileId          tech = techOpenTechFile("design", "tech.db", "w");

dbLayer             layers[] = { 6, 8 };
dbPurpose           purposes[] = { 1001, 1002 };
dbLayer             snapLayerNum = 8;
dbPurpose           snapPurposeNum = 5;
Boolean            isHorizontal = TRUE;
dbDistance          offset = 30;
dbDistance          period = 20;

const unsigned int  num_snapLayers = 1;
dbSnappingLayer     *snapLayers = dbAllocSnappingLayerArray(num_snapLayers);

snapLayers[0].layerNum = snapLayerNum;
snapLayers[0].purposes = calloc(1, dbPurpose);
snapLayers[0].numPurposes = 1;
snapLayers[0].purposes[0] = snapPurposeNum;

constString          wsspDefNames[] = { "wsspdef1", "wsspdef2" };
const unsigned int   num_wsspDefs = ARRAY_SIZE(wsspDefNames);
dbWidthSpacingSnapPatternDefId wsspDefs[num_wsspDefs];

constString          defaultWSP = "1X2X";

for (int i = 0; i < num_wsspDefs; i++) {
    wsspDefs[i] = dbCreateWidthSpacingSnapPatternDef(cellview,
                                                    wsspDefNames[i],
                                                    layers[i], purposes[i],
                                                    period, isHorizontal,
                                                    offset, snapLayers, 1,
                                                    &wspNames[0], num_wsps,
                                                    &wspgNames[0], num_wspgs,
                                                    defaultWSP);

    assert(wsspDefs[i]);
}

// Examples for the getter functions.
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
cout << "Sample widthSpacingSnapPatternDef dump: " << endl;
cout << "  name: "
<< dbGetWidthSpacingSnapPatternDefName(wsspDefs[0]) << endl;
cout << "  layer: "
<< dbGetWidthSpacingSnapPatternDefLayer(wsspDefs[0]) << endl;
cout << "  purpose: "
<< dbGetWidthSpacingSnapPatternDefPurpose(wsspDefs[0]) << endl;
cout << "  period: "
<< dbGetWidthSpacingSnapPatternDefPeriod(wsspDefs[0]) << endl;
cout << "  offset: "
<< dbGetWidthSpacingSnapPatternDefOffset(wsspDefs[0]) << endl;
cout << "  horizontal: "
<< (bool) dbIsWidthSpacingSnapPatternDefHorizontal(wsspDefs[0]) << endl;
cout << "  default active pattern: "
<< dbGetWidthSpacingSnapPatternDefDefaultActiveName(wsspDefs[0])
<< endl;

dbSnappingLayer*    tmp_snapLayers = NULL;

// Test dbGetWidthSpacingSnapPatternDefSnappingLayers().
dbGetWidthSpacingSnapPatternDefSnappingLayers(wsspDefs[0], &tmp_num,
                                              &tmp_snapLayers);

assert(tmp_num == num_snapLayers);
cout << "  snapping layers:" << endl;

for (int i = 0; i < tmp_num; i++) {
    cout << "    layer: " << tmp_snapLayers[i].layerNum << " purposes:";

    for (int j = 0; j < tmp_snapLayers[i].numPurposes; j++)
        cout << " " << tmp_snapLayers[i].purposes[j];

    cout << endl;
}

cout << endl;
dbFreeSnappingLayerArray(tmp_num, tmp_snapLayers);

// Test dbGetWidthSpacingSnapPatternDefPatternNames().
dbGetWidthSpacingSnapPatternDefPatternNames(wsspDefs[0], &tmp_num,
                                              &tmp_names);

assert(tmp_num == num_wsps);
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
cout << "  patterns:";

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);

// Test dbGetWidthSpacingSnapPatternDefPatternGroupNames().
dbGetWidthSpacingSnapPatternDefPatternGroupNames(wsspDefs[0], &tmp_num,
                                                  &tmp_names);

assert(tmp_num == num_wspgs);
cout << "  pattern groups:";

for (int i = 0; i < tmp_num; i++)
    cout << " " << tmp_names[i];

cout << endl;

free(tmp_names);

// Examples for the generator functions.
gen_state = dbStartGenWidthSpacingSnapPatternDef(cellview);
dbWidthSpacingSnapPatternDefId tmp_wsspdef;

cout << "Snap pattern definition names returned by the generator: ";

while (dbGenWidthSpacingSnapPatternDef(gen_state, &tmp_wsspdef))
    cout << " " << dbGetWidthSpacingSnapPatternDefName(tmp_wsspdef);

cout << endl << endl;
dbStopGen(gen_state);

// Examples for the find and delete functions.
dbLayer      tmp_layer = 10;
dbPurpose    tmp_purpose = 1003;
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
assert(dbFindWidthSpacingSnapPatternDefByName(cellview, "tmp_wsspdef")
      == dbcNullWidthSpacingSnapPatternDefId);
tmp_wsspdef = dbCreateWidthSpacingSnapPatternDef(cellview,
                                                  "tmp_wsspdef",
                                                  tmp_layer, tmp_purpose,
                                                  period, isHorizontal,
                                                  offset, snapLayers, 1,
                                                  NULL, 0,
                                                  NULL, 0,
                                                  defaultWSP);
assert(dbFindWidthSpacingSnapPatternDefByName(cellview, "tmp_wsspdef"));
assert(dbDeleteWidthSpacingSnapPatternDef(tmp_wsspdef));
assert(dbFindWidthSpacingSnapPatternDefByName(cellview, "tmp_wsspdef")
      == dbcNullWidthSpacingSnapPatternDefId);

tmp_wsspdef = dbCreateWidthSpacingSnapPatternDef(cellview,
                                                  "tmp_wsspdef",
                                                  tmp_layer, tmp_purpose,
                                                  period, isHorizontal,
                                                  offset, snapLayers, 1,
                                                  NULL, 0,
                                                  NULL, 0,
                                                  defaultWSP);
assert(dbDeleteWidthSpacingSnapPatternDef(tmp_wsspdef));

// *** RELATED SNAP PATTERNS API EXAMPLES ***

// Create the test data.
const unsigned int      num_rspSpecs = 1;
dbRelatedSnapPatternsSpec *rspSpecs =
dbAllocRelatedSnapPatternsSpecArray(num_rspSpecs);

constString              *wspNames2 = calloc(num_wsps, constString);
constString              *wspgNames2 = calloc(num_wspgs, constString);

memcpy(wspNames2, wspNames, num_wsps * sizeof(constString));
memcpy(wspgNames2, wspgNames, num_wspgs * sizeof(constString));

rspSpecs[0] = { (String) wsspDefNames[0], (String *) wspNames2, num_wsps,
               (String *) wspgNames2, num_wspgs };
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
// Create invalid specs.
const unsigned int      num_rspSpecs_invalid = 3;
dbRelatedSnapPatternsSpec *rspSpecs_invalid =
dbAllocRelatedSnapPatternsSpecArray(num_rspSpecs_invalid);
constString             *wspNames_invalid = calloc(1,
                                                    constString);

constString             *wspgNames_invalid = calloc(1,
                                                    constString);

wspNames_invalid[0] = "bad_wsp";
wspgNames_invalid[0] = "bad_wspg";

// Nonexisting widthSpacingPattern name.
rspSpecs_invalid[0] = { (String) wsspDefNames[0],
                        (String *) wspNames_invalid, 1, NULL, 0 };
// Nonexisting widthSpacingPatternGroup name.
rspSpecs_invalid[1] = { (String) wsspDefNames[0], NULL, 0,
                        (String *) wspgNames_invalid, 1};
// Nonexisting widthSpacingSnapPatternDef name.
rspSpecs_invalid[2] = { "bad_wsspdef", NULL, 0, NULL, 0};

// These commands will fail due to invalid relatedSnapPatternsSpecs.
for (int i = 0; i < num_rspSpecs_invalid; i++)
    assert(dbCreateRelatedSnapPatterns(cellview, "bad_rsp",
                                       rspSpecs_invalid + i, 1)
           == dbcNullRelatedSnapPatternsId);

// This command generates a valid relatedSnapPatterns object.
dbRelatedSnapPatternsId rsp = dbCreateRelatedSnapPatterns(cellview,
                                                           "rsp0",
                                                           rspSpecs,
                                                           num_rspSpecs);

assert(rsp);

// End of the examples section.
// The remaining auxiliary objects should be deallocated.
dbFreeSnappingLayerArray(1, snapLayers);
dbFreeWidthSpacingPatternSpecArray(num_wspSpecs, wspSpecs);
```

Cadence Integrators Toolkit Database Reference

Width Spacing Pattern Functions

```
dbFreeWidthSpacingPatternSpecArray(num_wspSpecs_invalid, wspSpecs_invalid);  
dbFreeRelatedSnapPatternsSpecArray(num_rspSpecs, rspSpecs);
```

Figure Set Functions

This chapter includes the following 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 an 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 not database objects and are not 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 on the disk, you need to create a corresponding FigSetGroup. For information about FigSetGroups, see [FigSetGroup Functions](#).

FigSet C APIs are used to create, find, move, copy, append, prepend, clone, clear, and delete objects from FigSets; to access FigSet parameters; and to iterate over the objects stored in a cellview.

To learn how to use FigSet and FigSetGroup functions, see [Example- Using the FigSet and FigSetGroup Code](#).

Supported functions include:

- [txAppendFig](#)
- [txAppendFigSet](#)
- [txClearFigSet](#)
- [txCloneFigSet](#)
- [txConcatFigSet](#)
- [txCopyFigSet](#)
- [txCreateFigSet](#)
- [txDeleteFigSet](#)
- [txFindFigSet](#)
- [txGetCellViewFigSets](#)
- [txGetFigSetBBox](#)
- [txGetFigSetCellView](#)
- [txGetFigSetEntry](#)
- [txGetFigSetLength](#)

Cadence Integrators Toolkit Database Reference

Figure Set Functions

- [txGetFigSetName](#)
- [txIsFigSet](#)
- [txMakeFigSet](#)
- [txMoveFigSet](#)
- [txPrependFig](#)
- [txPrependFigSet](#)
- [txRemoveFig](#)
- [txRemoveFigSet](#)
- [txRemoveFigSetEntry](#)

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txAppendFig

Boolean

```
txAppendFig(  
    txFigSetId fsId,  
    dbFigId    figId  
);
```

Description

Adds the specified figure as a member to the end of the given FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.
<i>figId</i>	ID of a figure.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

txAppendFigSet

Boolean

```
txAppendFigSet(  
    txFigSetId fsId,  
    txFigSetId fsId2  
);
```

Description

Adds the second FigSet as a member to the end of the first FigSet.

Arguments

<i>fsId</i>	ID of the first FigSet.
<i>fsId2</i>	ID of the second FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

txClearFigSet

Boolean

```
txClearFigSet(  
    txFigSetId fsId  
);
```

Description

Clears the contents of the specified FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

txCloneFigSet

Boolean

```
txCloneFigSet(  
    txFigSetId fsId,  
    txFigSetId fsId2  
);
```

Description

Clears the contents of the first FigSet and then, copies the contents of the second FigSet to the first FigSet. The name for the first FigSet is copied from the name of the second FigSet.

Arguments

<i>fsId</i>	ID of the first FigSet.
<i>fsId2</i>	ID of the second FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txConcatFigSet

Boolean

```
txConcatFigSet(  
    txFigSetId fsId,  
    txFigSetId fsId2  
);
```

Description

Concatenates the contents of the second FigSet to the end of the first FigSet. The contents of the second FigSet remain unchanged.

Arguments

<i>fsId</i>	ID of the first FigSet.
<i>fsId2</i>	ID of the second FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txCopyFigSet

```
txFigSetId  
txCopyFigSet(  
    txFigSetId  srcFigSetId,  
    dbTransform transform  
);
```

Description

Creates copies of each figure in the specified FigSet with the specified *transform*. A new FigSet containing the copied figures is returned. The copies are named automatically by using the convention *FS_n*, where *n* is a generated number that makes the name unique. So, the first FigSet name will be *FS_0*, then *FS_1* and so on.

Arguments

<i>srcFigSetId</i>	ID of the source FigSet.
<i>transform</i>	The transformation object that is applied while copying figures in the given FigSet.

Return Value

<i>txFigSetId</i>	ID of the new FigSet.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txCreateFigSet

```
txFigSetId  
txCreateFigSet(  
    dbCellViewId cvId,  
    constString name  
);
```

Description

Creates an empty FigSet in the specified cellview. If the *name* argument is specified, the FigSet is created with the specified name. If the *name* argument is not specified (NULL), the FigSet is named automatically. Automatic naming is done using the convention *FS_n*, where *n* is a generated number that makes the name unique. So, the first FigSet name will be *FS_0*, then *FS_1* and so on.

Arguments

<i>cvId</i>	ID of the cellview where the new empty FigSet is created.
<i>name</i>	Name of the FigSet to be created.

Return Value

<i>txFigSetId</i>	ID of the new FigSet.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txDeleteFigSet

Boolean

```
txDeleteFigSet(  
    txFigSetId fsId,  
    Boolean    deep  
);
```

Description

Deletes the specified FigSet, but the objects that it contains are not deleted. If the FigSet is a member of another FigSet, it is removed from that FigSet before it is deleted. Deleting the FigSet frees its associated memory immediately.

Arguments

fsId

ID of a FigSet.

deep

Specifies whether referenced FigSets should be deleted or not.

TRUE, if the referenced FigSets are to be deleted.

FALSE, if the referenced FigSets are not to be deleted.

Return Value

Boolean

TRUE, if the operation succeeded.

FALSE, if the operation failed.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txFindFigSet

```
txFigSetId  
txFindFigSet  
    (dbCellViewId cvId,  
     constString name  
    );
```

Description

Searches the cellview for a FigSet with the specified name. When searching for the FigSet, a matching FigSetGroup is also searched and, if necessary, a corresponding FigSet is resurrected automatically from it.

Arguments

<i>cvId</i>	ID of the cellview in which search is performed.
<i>name</i>	Name of the FigSet to search.

Return Value

<i>txFigSetId</i>	ID of the FigSet, if found. Returns <code>NULL</code> , if the specified FigSet is not found.

txGetCellViewFigSets

```
txFigSetId *  
txGetCellViewFigSets(  
    dbCellViewId cvId  
);
```

Description

Returns an array of FigSets contained in the given design. The array is terminated by `txcNullFigSetId`.

Arguments

<i>cvId</i>	ID of the cellview in which the FigSets are searched.

Return Value

<i>txFigSetId</i>	Array containing IDs of all FigSets in the specified cellview.

txGetFigSetBBox

dbBBox *

```
txGetFigSetBBox(
    txFigSetId    fsId,
    unsigned int  numIncludeLayers,
    dbLayer       *includeLayers,
    unsigned int  numExcludeLayers,
    dbLayer       *excludeLayers,
    unsigned int  numIncludeLPPs,
    txLPP         *includeLPPs,
    unsigned int  numExcludeLPPs,
    txLPP         *excludeLPPs,
    unsigned int  numIncludeTypes,
    dbType        *includeTypes,
    unsigned int  numExcludeTypes,
    dbType        *excludeTypes
);
```

Description

Calculates the bounding box for the objects in the specified FigSet and returns a pointer to the resulting bounding box. Optionally, included and excluded layers, LPPs, and object types can also be specified.

Arguments

<i>fsId</i>	ID of the FigSet in which the bounding box of the objects is calculated.
<i>numIncludeLayers</i>	Number of layers in the array of layers to be included.
<i>includeLayers</i>	Array of layers to be included.
<i>numExcludeLayers</i>	Number of layers in the array of layers to be excluded.
<i>excludeLayers</i>	Array of layers to be excluded.
<i>numIncludeLPPs</i>	Number of LPPs in the array of LPPs to be included.
<i>includeLPPs</i>	Array of LPPs to be included.
<i>numExcludeLPPs</i>	Number of LPPs in the array of LPPs to be excluded.
<i>excludeLPPs</i>	Array of LPPs to be excluded.
<i>numIncludeTypes</i>	Number of object types in the array of object types to be included.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

<i>includeTypes</i>	Array of object types to be included.
<i>numExcludeTypes</i>	Number of object types in the array of object types to be excluded.
<i>excludeTypes</i>	Array of object types to be excluded.

Return Value

<i>dbBBox</i>	Pointer to the resulting bounding box. Returns <code>NULL</code> , if there is any error in validating the arguments.

txGetFigSetCellView

```
dbCellViewId  
txGetFigSetCellView(  
    txFigSetId fsId  
);
```

Description

Returns the ID of the parent cellview for the specified FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.

Return Value

<i>dbCellViewId</i>	ID of the parent cellview.

txGetFigSetEntry

Boolean

```
txGetFigSetEntry(  
    txFigSetId    fsId,  
    unsigned int  index,  
    txFigSetEntry *entry  
);
```

Description

Retrieves the FigSet element at the specified `index` location and places it at the memory area specified using `entry`.

Arguments

<i>fsId</i>	ID of a FigSet.
<i>index</i>	Index of an entry in the array of elements of <i>fsId</i> .
<i>entry</i>	Pointer to the memory area where the FigSet entry will be placed.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

txGetFigSetLength

```
unsigned int  
txGetFigSetLength(  
    txFigSetId fsId  
);
```

Description

Returns the number of objects in the specified FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.

Return Value

<i>unsigned int</i>	Number of objects in the specified FigSet.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txGetFigSetName

String

```
txGetFigSetName(  
    txFigSetId fsId  
);
```

Description

Returns the name for the specified FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.

Return Value

<i>String</i>	Name of the specified FigSet.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txIsFigSet

```
Boolean  
txIsFigSet(  
    txId id  
);
```

Description

Determines whether the specified database object is a FigSet object.

Arguments

<i>id</i>	ID of a database object.

Return Value

<i>Boolean</i>	TRUE, if the specified object is a FigSet. FALSE, if the specified object is not a FigSet, or if an error occurred.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txMakeFigSet

```
txFigSetId  
txMakeFigSet (  
    dbFigSetGroupId  figSetGroupId  
);
```

Description

Creates a FigSet from the given FigSetGroup and returns the new FigSet. The FigSet will be created if it does not exist; if it already exists, its contents will be replaced.

Arguments

<i>figSetGroupId</i>	ID of a FigSetGroup.

Return Value

<i>txFigSetId</i>	ID of the new FigSet.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txMoveFigSet

Boolean

```
txMoveFigSet(  
    txFigSetId  figSetId,  
    dbTransform transform  
);
```

Description

Moves each figure in the specified FigSet by the specified *transform*.

Arguments

<i>figSetId</i>	ID of a FigSet.
<i>transform</i>	Transformation object that is applied while moving the figures in the specified FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txPrependFig

Boolean

```
txPrependFig(  
    txFigSetId fsId,  
    dbFigId    figId  
);
```

Description

Adds the specified figure as a member at the beginning of the specified FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.
<i>figId</i>	ID of a figure.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

txPrependFigSet

Boolean

```
txPrependFigSet(  
    txFigSetId fsId,  
    txFigSetId fsId2  
);
```

Description

Adds the second FigSet as a member to the beginning of the first FigSet.

Arguments

<i>fsId</i>	ID of the first FigSet.
<i>fsId2</i>	ID of the second FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txRemoveFig

Boolean

```
txRemoveFig(  
    txFigSetId    fsId,  
    dbFigId       figId  
);
```

Description

Removes the specified figure from the specified FigSet .

Arguments

<i>fsId</i>	ID of a FigSet.
<i>figId</i>	ID of a figure.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

txRemoveFigSet

Boolean

```
txRemoveFigSet (
    txFigSetId    fsId,
    txFigSetId    fsId2
);
```

Description

Removes the second FigSet from the first FigSet.

Arguments

<i>fsId</i>	ID of the first FigSet.
<i>fsId2</i>	ID of the second FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded.
	FALSE, if the operation failed.

txRemoveFigSetEntry

Boolean

```
txRemoveFigSetEntry(  
    txFigSetId  fsId,  
    unsigned int index  
);
```

Description

Removes the entry at the specified *index* from the specified FigSet.

Arguments

<i>fsId</i>	ID of a FigSet.
<i>index</i>	Index of an entry in the array of elements of the specified FigSet.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeded. FALSE, if the operation failed.

FigSetGroup Functions

A FigSetGroup is a method to store the contents of a FigSet in a database for use in different sessions.

FigSetGroup C APIs are used to create, delete, access, manage, and search FigSetGroup objects.

To learn how to use FigSet and FigSetGroup functions, see [Example- Using the FigSet and FigSetGroup Code](#).

Supported functions include:

- [dbDeleteFigSetGroup](#)
- [dbFindFigSetGroup](#)
- [dbFindFigSetGroupByName](#)
- [dbGenFigSetGroups](#)
- [dbGetFigSetGroupName](#)
- [dbIsFigSetGroup](#)
- [dbMakeFigSetGroup](#)
- [dbStartGenFigSetGroups](#)

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbDeleteFigSetGroup

Boolean

```
dbDeleteFigSetGroup(  
    dbFigSetGroupId    figSetGroupId,  
    );
```

Description

Deletes the specified FigSetGroup and the hierarchy of FigSetGroups that might exist in it.

Arguments

<i>figSetGroupId</i>	ID of the FigSetGroup to be deleted.

Return Value

<i>Boolean</i>	TRUE, if the operation succeeds. FALSE, if the procedure fails.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbFindFigSetGroup

```
dbFigSetGroupId  
dbFindFigSetGroup(  
    dbCellViewId    cvId,  
    txFigSetId      figSetId  
);
```

Description

Searches the specified cellview for a FigSetGroup that has the same name as the specified FigSet.

Arguments

<i>cvId</i>	ID of the cellview in which the search is performed.
<i>figSetId</i>	ID of the FigSet for which the matching FigSetGroup is being searched.

Return Value

<i>dbFigSetGroupId</i>	ID of the found FigSetGroup.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbFindFigSetGroupName

```
dbFigSetGroupId  
dbFindFigSetGroupName(  
    dbCellViewId    cvId,  
    constString     name  
);
```

Description

Searches the specified cellview for a FigSetGroup with the specified name.

Arguments

<i>cvId</i>	ID of the cellview in which the search is performed.
<i>name</i>	Name of the FigSetGroup for which the search is performed.

Return Value

<i>dbFigSetGroupId</i>	ID of the FigSetGroup. Returns <code>dbcNullId</code> , if the named FigSetGroup does not exist.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbGenFigSetGroups

Boolean

```
dbGenFigSetGroups (  
    dbGenStateId  pState,  
    dbFigSetGroup *figSetGroupId  
);
```

Description

Retrieves the next FigSetGroup ID from the generator with the given `dbGenStateId`. The FigSetGroup ID is returned in the `figSetGroupId` argument.

Arguments

<i>pState</i>	State ID for the generator.
<i>figSetGroupId</i>	ID of the next FigSetGroup produced by the generator. This value is undefined if <code>FALSE</code> is returned.

Return Value

<i>Boolean</i>	<code>TRUE</code> , if the next FigSetGroup ID is successfully generated. <code>FALSE</code> , if the list is exhausted or an error occurs.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbGetFigSetGroupName

String

```
dbGetFigSetGroupName (  
    dbFigSetGroupId figSetGroupId  
);
```

Description

Returns the name for the specified FigSetGroup object.

Arguments

<i>figSetGroupId</i>	ID of a FigSetGroup object.

Return Value

<i>String</i>	Name of the FigSetGroup object.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbIsFigSetGroup

Boolean

```
dbIsFigSetGroup(  
    dbId    id  
);
```

Description

Indicates whether the specified `dbId` is a `FigSetGroup` object.

Arguments

<i>id</i>	ID of the database object to be checked.

Return Value

<i>Boolean</i>	TRUE, if the specified <code>dbId</code> is a <code>FigSetGroup</code> object. FALSE, if the specified <code>dbId</code> is not a <code>FigSetGroup</code> object or in case of failure.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbMakeFigSetGroup

```
dbFigSetGroupId  
dbMakeFigSetGroup(  
    dbCellViewId cvId,  
    txFigSetId   figSetId  
);
```

Description

Makes a FigSetGroup in the specified cellview that is associated with the specified FigSet and populates it with the contents of the FigSet. If a FigSetGroup with the same name as the specified FigSet does not exist, it is created and assigned the name of the FigSet. If such a FigSetGroup exists, its contents are cleared and replaced with the contents of the FigSet.

Arguments

<i>cvId</i>	ID of a cellview.
<i>figSetId</i>	ID of a FigSet.

Return Value

<i>dbFigSetGroupId</i>	ID of the created FigSetGroup.

Cadence Integrators Toolkit Database Reference

Figure Set Functions

dbStartGenFigSetGroups

```
dbGenStateId  
dbStartGenFigSetGroups (  
    dbCellViewId    cellViewId  
);
```

Description

Initializes a generator for all FigSetGroup objects in the specified cellview. A generator state is returned, which is used with the [dbGenFigSetGroups](#) function.

Arguments

<i>cellViewId</i>	ID of the cellview containing the FigSetGroups to be generated.

Return Value

<i>dbGenStateId</i>	State ID for the generator.

Example- Using the FigSet and FigSetGroup Code

The following example illustrates how to use the FigSet and FigSetGroup APIs.

```
void print(txFigSetId figSet) {
    txFigSetEntry entry;

    printf(" %s=> [", txGetFigSetName(figSet));

    for (unsigned int i = 0; i < txGetFigSetLength(figSet); i++) {
        txGetFigSetEntry(figSet, i, &entry);

        if (entry.type == txcFigureSetFig) {
            printf("rect%d ", dbGetShapeRec(entry.data.figId)->layer);
        } else {
            print(entry.data.figSetId);
        }
    }

    printf("] ");
}

void example() {
    dbBBox          bbox0;
    dbBBox          bbox1;

    dbFillBBox(&bbox0, 0, 0, 100, 100);
    dbFillBBox(&bbox1, -100, -100, 0, 0);

    dbCellViewId    cvId = dbOpenCellViewByType(libName, "rect", "layout",
                                                "maskLayout", "r", dbcNullCellViewId);
    dbRectId        rect0  = dbCreateRect(cvId, 0, dbcDrawingPurpose, &bbox0);
    dbRectId        rect1  = dbCreateRect(cvId, 1, dbcDrawingPurpose, &bbox1);

    // Create a FigSet hierarchy
    txFigSetId      figSetTop = txCreateFigSet(cvId, "topFigSet");
    txFigSetId      figSet0 = txCreateFigSet(cvId, NULL); // Auto-named ->"FS_0"
    txFigSetId      figSet1 = txCreateFigSet(cvId, NULL); // Auto-named ->"FS_1"

    txAppendFig(figSet0, rect0);
}
```

Cadence Integrators Toolkit Database Reference

Figure Set Functions

```
txAppendFig(figSet1, rect1);
txAppendFigSet(figSetTop, figSet1);
txPrependFigSet(figSetTop, figSet0);
txPrependFig(figSetTop, rect1);

printf("Initial state\n");
print(figSetTop);
printf("\n");

txFigSetId      clone = txCreateFigSet(cvId, "clone");

if (txCloneFigSet(clone, figSetTop)) {
    printf("Clone\n");
    print(clone);
    printf("\n");
}

if (txClearFigSet(clone)) {
    printf("After clearing the contents of the clone\n");
    print(clone);
    printf("\n");
}

dbTransform      transform;

transform.offset.x = 50000;
transform.offset.y = 0;
transform.orient   = dbcR0;
transform.mag      = 1.;

txFigSetId      copy = txCopyFigSet(figSetTop, transform);

printf("Copy\n");
print(copy);
printf("\n");

dbBBox          *figSetBBox = txGetFigSetBBox(figSetTop, 0, NULL, 0, NULL, 0,
NULL, 0, NULL, 0, NULL, 0, NULL);
dbLayer         layers[1] = {0};
```

Cadence Integrators Toolkit Database Reference

Figure Set Functions

```
dbLayer          layers2[2] = {0, 1};

printf("BBox of figSetTop : (%d, %d), (%d, %d)\n",
      figSetBBox->lowerLeft.x,  figSetBBox->lowerLeft.y,
      figSetBBox->upperRight.x, figSetBBox->upperRight.y);

figSetBBox = txGetFigSetBBox(figSetTop, 1, layers, 0, NULL, 0, NULL,
                             0, NULL, 0, NULL, 0, NULL);
printf("BBox of figSetTop including only Layer 0: (%d, %d), (%d, %d)\n",
      figSetBBox->lowerLeft.x,  figSetBBox->lowerLeft.y,
      figSetBBox->upperRight.x, figSetBBox->upperRight.y);

figSetBBox = txGetFigSetBBox(figSetTop, 0, NULL, 1, layers, 0, NULL,
                             0, NULL, 0, NULL, 0, NULL);
printf("BBox of figSetTop excluding only Layer 0: (%d, %d), (%d, %d)\n",
      figSetBBox->lowerLeft.x,  figSetBBox->lowerLeft.y,
      figSetBBox->upperRight.x, figSetBBox->upperRight.y);

figSetBBox = txGetFigSetBBox(figSetTop, 0, NULL, 2, layers2, 0, NULL,
                             0, NULL, 0, NULL, 0, NULL);
printf("BBox of figSetTop excluding Layer 0 and 1: (%d, %d), (%d, %d)\n",
      figSetBBox->lowerLeft.x,  figSetBBox->lowerLeft.y,
      figSetBBox->upperRight.x, figSetBBox->upperRight.y);

// Make a figSetGrouP and vice-versa.
dbFigSetGroupId figSetGrouTop0 = dbMakeFigSetGroup(cvId, figSetTop);

if (txClearFigSet(figSetTop)) {
    txMakeFigSet(figSetGrouTop0);
    printf("After resurrecting the top FigSet from a FigSet group\n");
    print(figSetTop);
    printf("\n");
}

// Iterate on all the existing FigSetGroups and print their name.
printf("Existing FigSet group : ");

dbGenStateId    genState = dbStartGenFigSetGroups(cvId);
dbFigSetGroupId figSetGroupId;
```

Cadence Integrators Toolkit Database Reference

Figure Set Functions

```
while (dbGenFigSetGroups(genState, &figSetGroupId)) {  
    printf("%s ", dbGetFigSetName(figSetGroupId));  
}  
}
```

The example above generates the following output:

Initial state

```
topFigSet=> [rect1 FS_0=> [rect0 ] FS_1=> [rect1 ] ]
```

Clone

```
clone=> [rect1 FS_0=> [rect0 ] FS_1=> [rect1 ] ]
```

Clear the contents of the clone

```
clone=> []
```

Copy

```
FS_2=> [rect1 FS_3=> [rect0 ] FS_4=> [rect1 ] ]
```

BBox of figSetTop : (-100, -100), (100, 100)

BBox of figSetTop including only Layer 0: (0, 0), (100, 100)

BBox of figSetTop excluding only Layer 0: (-100, -100), (0, 0)

BBox of figSetTop excluding Layer 0 and 1: (0, 0), (0, 0)

After resurrecting the top FigSet from a FigSetGroup

```
topFigSet=> [rect1 FS_0=> [rect0 ] FS_1=> [rect1 ] ]
```

Existing FigSetGroup : topFigSet FS_0 FS_1

Dictionary of Basic CDBA Terms

This appendix contains a compilation of basic terms used for naming functions, constants, and variables. If an official abbreviation exists for the term, it is listed in parentheses after the term.

The terms are categorized here into

- General Terms on page 2144
- Terms Involving Actions on page 2144
 - Creating and Deleting Objects on page 2145
 - Actions Using Attributes on page 2145
 - Testing and Bookkeeping on page 2146
 - Actions Involving Input and Output for Cellviews on page 2146
 - Actions on Collections of Objects on page 2146
- Adjectival Terms on page 2147
- Alphabetical List of Abbreviations on page 2147.

Note: This dictionary is not yet complete.

General Terms

The following are general terms used in naming functions, constants, and variables.

Object name	Abbreviation	Description
Bounding Box	BBox	Smallest rectangle that completely contains an object
Coordinate	Coord	Distance along the horizontal (x) and vertical (y) axes (2-dimensions)
Hierarchy	Hier	Design levels built by creating an instance of one cell into another cell
Identifier	Id	Variable or structure that describes a unique occurrence of a database object
Layer		Three dimensions
Library	Lib	Collection of cells
Path		Collection of libraries containing cells
Point		Location specified by one x and one y coordinate
Purpose		What an object does
Layer-Purpose	LP	Combination of layer and purpose. Separately, a layer and a purpose are concepts, not physical entities. A layer and a purpose combined, however, make a physical entity called a "layer-purpose pair." See Layer-Purpose (LP) Pair Functions on page 1274 for a more detailed description.

Terms Involving Actions

The following terms refer to the following actions on database objects:

- Creating and deleting objects
- Setting and getting attribute values for objects
- Testing for Boolean values and performs bookkeeping actions for functions

Cadence Integrators Toolkit Database Reference

Dictionary of Basic CDBA Terms

- Performs input and output on cellviews

Creating and Deleting Objects

The terms for creating and deleting database objects are described as follows. The abbreviation for a term is given, if any exists.

Action	Abbreviation	Description
Copy		Copies an object or data structure
Create		Creates an object
Delete		Deletes an object
Initialize	Init	Sets initial values
Make		Finds an object if it already exists, otherwise make creates it.
Move		Moves an object

Actions Using Attributes

The terms for actions using attributes are described as follows.

Action	Description
Fill	Fills in a template with attributes
Get	Gets attributes of an object
Set	Sets attributes of an object

Cadence Integrators Toolkit Database Reference

Dictionary of Basic CDBA Terms

Testing and Bookkeeping

The terms used to test for Boolean values and perform bookkeeping actions for functions are described as follows.

Action	Abbreviation	Description
Count		Counts the number of objects in a collection
Has		Tests if a relationship exists and returns TRUE if the relationship exists
Is		Tests for an object type or a specific condition and returns TRUE or FALSE
Register	Reg	Registers a handler or daemon function
Unregister	Unreg	Unregisters a handler or daemon function

Actions Involving Input and Output for Cellviews

The terms for actions involving input and output for cellviews are described as follows.

Action	Description
Open	Opens a cellview for reading or writing
Close	Closes a cellview
Save	Saves a cellview on disk
Purge	Removes a cellview from virtual memory

Actions on Collections of Objects

The terms used in actions on collections of objects are described as follows. The abbreviation for a term is given, if any exists.

Action	Abbreviation	Description
Add		Adds something to a set or relationship

Cadence Integrators Toolkit Database Reference

Dictionary of Basic CDBA Terms

Action	Abbreviation	Description
Subtract	Sub	Subtracts something from a set or relationship
Generate	Gen	Generates objects in a set
Start		Starts an ongoing event, usually to activate a generator
Stop		Stops an ongoing event, usually to inactivate a generator
Select		Selects objects that meet given criteria
Find		Finds an object with the specified attribute

Adjectival Terms

The following adjectival terms indicating quantity, degree, or condition are described. The abbreviation is given, if one exists.

Adjectives	Abbreviation	Description
All		Everything
Equal		Equal in value or pointing to the same object
Minimum	Min	Smallest value
Maximum	Max	Largest value
Permanent	Perm	Object persists for the lifetime of the process
Temporary	Temp	Object is volatile

Alphabetical List of Abbreviations

The following is an alphabetical list of abbreviations for the terms listed in this appendix.

Abbreviation	Full name
bBox	bounding box
conn	connection

Cadence Integrators Toolkit Database Reference

Dictionary of Basic CDBA Terms

Abbreviation	Full name
CCW	counter-clockwise
CW	clockwise
db	database
det	determinant
dm	design data management
dBox	dialog box
DBU	database units
env	environment
fig	figure
GC	garbage collection
gen	generate
hier	hierarchy
id	identifier
ident	identity
info	information
inst	instance
iter	iterated
lib	library
LP	layer-purpose
mem	member
msg	message
neg	negative
num	number
obj	object
perm	permanent
plac	placement
pos	positive

Cadence Integrators Toolkit Database Reference

Dictionary of Basic CDBA Terms

Abbreviation	Full name
prop	property
prog	programming
rect	rectangle
rev	revision
rot	rotation
temp	temporary
term	terminal
UU	user units
warn	warning