



Allegro®

PCB and Package User Guide:

SKILL Reference

Series 200 and 600

Product Version 15.5
July 2005



© 1999-2005 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

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

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) 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 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. 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. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

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

Contents

<u>Alphabetical List of Functions</u>	23
---------------------------------------	----

<u>Before You Start</u>	35
-------------------------	----

<u>About This Manual</u>	35
--------------------------	----

<u>Prerequisites</u>	36
----------------------	----

<u>Command Syntax Conventions</u>	36
-----------------------------------	----

<u>Referencing Objects by Name</u>	37
------------------------------------	----

<u>Finding Information in This Manual</u>	37
---	----

<u>Other Sources of Information</u>	44
-------------------------------------	----

1

<u>Introduction to PCB Editor SKILL Functions</u>	49
---	----

<u>Overview</u>	49
-----------------	----

<u>AXL-SKILL in PCB Editor</u>	49
--------------------------------	----

<u>AXL-SKILL Database</u>	52
---------------------------	----

2

<u>The PCB Editor Database User Model</u>	61
---	----

<u>Overview</u>	61
-----------------	----

<u>Description of Database Objects</u>	62
--	----

<u>Figure Database Types</u>	67
------------------------------	----

<u>Logical Database Types</u>	81
-------------------------------	----

<u>Property Dictionary Database Types</u>	87
---	----

<u>Parameter Database Types</u>	89
---------------------------------	----

3

<u>Database Create Functions</u>	99
----------------------------------	----

<u>Overview</u>	99
-----------------	----

<u>Path Functions</u>	100
-----------------------	-----

Allegro PCB and Package User Guide: SKILL Reference

<u>axIPPathStart</u>	102
<u>axIPPathArcRadius</u>	104
<u>axIPPathArcAngle</u>	104
<u>axIPPathArcCenter</u>	104
<u>axIPPathLine</u>	108
<u>axIPPathGetWidth</u>	109
<u>axIPPathSegGetWidth</u>	110
<u>axIPPathGetPathSegs</u>	111
<u>axIPPathGetLastPathSeg</u>	112
<u>axIPPathSegGetEndPoint</u>	113
<u>axIPPathSegGetArcCenter</u>	114
<u>axIPPathSegGetArcClockwise</u>	115
<u>axIPPathStartCircle</u>	116
<u>axIDBCreatePath</u>	117
<u>axIDBCreateLine</u>	119
<u>axIDBCreateCircle</u>	121
<u>Create Shape Interface</u>	123
<u>axIDBCreateOpenShape</u>	125
<u>axIDBCreateCloseShape</u>	128
<u>axIDBActiveShape</u>	129
<u>axIDBCreateVoidCircle</u>	130
<u>axIDBCreateVoid</u>	131
<u>axIDBCreateShape</u>	132
<u>axIDBCreateRectangle</u>	134
<u>Nonpath DBCreate Functions</u>	136
<u>axIDBCreateExternalDRC</u>	136
<u>axIDBCreatePadStack</u>	139
<u>axIDBCreatePin</u>	143
<u>axIDBCreateSymbol</u>	146
<u>axIDBCreateSymbolSkeleton</u>	149
<u>axIDBCreateText</u>	153
<u>axIDBCreateVia</u>	155
<u>axIDBCreateSymbolAutosilk</u>	157
<u>Property Functions</u>	158
<u>axIDBCreatePropDictEntry</u>	158
<u>axIDBAddProp</u>	161

<u>Load Functions</u>	164
<u>axlLoadPadstack</u>	164
 <u>4</u>	
<u>Parameter Management Functions</u>	165
<u>Overview</u>	165
<u>axlColorGet</u>	166
<u>axlColorShadowGet</u>	168
<u>axlColorShadowSet</u>	169
<u>axlColorLoad</u>	171
<u>axlColorOnGet</u>	173
<u>axlColorOnSet</u>	175
<u>axlColorPriorityGet</u>	177
<u>axlColorPrioritySet</u>	179
<u>axlColorSave</u>	181
<u>axlColorSet</u>	182
<u>axlGetParam</u>	185
<u>axlSetParam</u>	186
<u>Database Layer Management</u>	187
<u>axlDBGetLayerType</u>	187
<u>axllsLayer</u>	188
<u>axllsVisibleLayer</u>	189
<u>axlLayerCreateCrossSection</u>	190
<u>axlLayerCreateNonConductor</u>	192
<u>axlLayerGet</u>	193
<u>axlVisibleDesign</u>	194
<u>axlVisibleGet</u>	196
<u>axlVisibleLayer</u>	198
<u>axlVisibleSet</u>	199
<u>axlConductorBottomLayer</u>	200
<u>axlConductorTopLayer</u>	201
<u>axlDBCreateFilmRec</u>	202
<u>axlSetPlaneType</u>	205

5

Selection and Find Functions	207
<u>Overview</u>	207
<u>Select Set Highlighting</u>	208
<u>Select Modes</u>	208
<u>Finding Objects by Name</u>	208
<u>Point Selection</u>	209
<u>Area Selection</u>	209
<u>Miscellaneous Select Functions</u>	209
<u>axlSelect—The General Select Function</u>	210
<u>Select Set Management</u>	210
<u>Find Filter Control</u>	210
Selection and Find Functions	212
<u>axlSingleSelectPoint</u>	212
<u>axlAddSelectPoint</u>	214
<u>axlSubSelectPoint</u>	215
<u>axlSingleSelectBox</u>	217
<u>axlAddSelectBox</u>	219
<u>axlSubSelectBox</u>	220
<u>axlAddSelectAll</u>	221
<u>axlSubSelectAll</u>	222
<u>axlSingleSelectName</u>	223
<u>axlAddSelectName</u>	225
<u>axlSubSelectName</u>	226
<u>axlSingleSelectObject</u>	227
<u>axlAddSelectObject</u>	228
<u>axlSubSelectObject</u>	229
<u>axlSelect</u>	230
<u>axlGetSelSet</u>	232
<u>axlGetSelSetCount</u>	234
<u>axlClearSelSet</u>	235
<u>axlGetFindFilter</u>	236
<u>axlSetFindFilter</u>	237
<u>axlAutoOpenFindFilter</u>	247
<u>axlOpenFindFilter</u>	248

<u>axlCloseFindFilter</u>	249
<u>axlFindFilterIsOpen</u>	250
<u>axlSelectByName</u>	251
<u>axlSelectByProperty</u>	256
 6	
<u>Interactive Edit Functions</u>	259
<u>Overview</u>	259
<u>AXL/SKILL Interactive Edit Functions</u>	260
<u>axlChangeWidth</u>	260
<u>axlDeleteObject</u>	262
<u>axlDBDeleteProp</u>	264
<u>axlGetLastEnterPoint</u>	267
<u>axlWindowBoxGet</u>	268
<u>axlWindowBoxSet</u>	269
<u>axlReplacePadstack</u>	270
<u>axlDeleteFillet</u>	271
<u>axlFillet</u>	272
<u>axlPurgePadstacks</u>	273
<u>axlShapeAutoVoid</u>	275
<u>axlShapeChangeDynamicType</u>	277
<u>axlShapeDeleteVoids</u>	279
<u>axlShapeDynamicUpdate</u>	281
<u>axlShapeRaisePriority</u>	282
<u>axlShovelItems</u>	284
<u>axlShoveSetParams</u>	285
<u>axlTransformObject</u>	288
 7	
<u>Database Read Functions</u>	293
<u>AXL-SKILL Database Read Functions</u>	293
<u>axlDBGetDesign</u>	294
<u>axlDBGetDrillPlating</u>	295
<u>axlIsDBIDType</u>	296
<u>axlDBGetAttachedText</u>	297

<u>axIDBGetPad</u>	299
<u>axIDBGetPropDictEntry</u>	301
<u>axIDBGetProperties</u>	302
<u>axIDBGetDesignUnits</u>	304
<u>axIDBRefreshId</u>	305
<u>axIDBGetLonelyBranches</u>	307
<u>axIDBGetConnect</u>	308
<u>axIDBGetManhattan</u>	309
<u>axIDBIsBondpad</u>	310
<u>axIDBIsBondwire</u>	311
<u>axIDBIsDiePad</u>	312
<u>axIDBIsFixed</u>	313
<u>axIDBIsPackagePin</u>	314
<u>axIDBIsPlatingbarPin</u>	315
<u>axIGetModuleInstanceDefinition</u>	316
<u>axIGetModuleInstanceLocation</u>	317
<u>axIGetModuleInstanceLogicMethod</u>	318
<u>axIGetModuleInstanceNetExceptions</u>	319
<u>axIIsDummyNet</u>	320
<u>axIIsLayerNegative</u>	321
<u>axIIsPinUnused</u>	322
<u>axIIsitFill</u>	323
<u>axIOK2Void</u>	324
<u>axIDBAssignNet</u>	325
<u>axIDBDynamicShapes</u>	326
<u>axIDBGetShapes</u>	327
<u>axIDBIsBondingWireLayer</u>	329
<u>axIDBTextBlockCompact</u>	330
 8	
<u>PCB Editor Interface Functions</u>	331
<u>Overview</u>	331
<u>AXL-SKILL Interface Function Examples</u>	331
<u>PCB Editor Interface Functions</u>	338
<u>axIClearDynamics</u>	338

<u>axlAddSimpleRbandDynamics</u>	339
<u>axlAddSimpleMoveDynamics</u>	341
<u>axlEnterPoint</u>	342
<u>axlEnterString</u>	343
<u>axlEnterAngle</u>	344
<u>axlCancelEnterFun</u>	345
<u>axlFinishEnterFun</u>	346
<u>axlGetDynamicsSegs</u>	347
<u>axlEnterBox</u>	348
<u>axlEnterPath</u>	350
<u>axlHighlightObject</u>	351
<u>axlDehighlightObject</u>	353
<u>axlMiniStatusLoad</u>	354
<u>axlDrawObject</u>	357
<u>axlDynamicsObject</u>	358
<u>axlEraseObject</u>	359
<u>axlControlRaise</u>	360
<u>axlEnterEvent</u>	361
<u>axlEventSetStartPopup</u>	365
<u>axlGetTrapBox</u>	367
<u>axlRatsnestBlank</u>	368
<u>axlRatsnestDisplay</u>	369
<u>axlShowObjectToFile</u>	370
<u>axlUICmdPopupSet</u>	371
<u>axlZoomToDbid</u>	372
<u>axlMakeDynamicsPath</u>	373

9

<u>PCB Editor Command Shell Functions</u>	375
<u>Overview</u>	375
<u>Command Shell Functions</u>	375
<u>axlGetVariable</u>	376
<u>axlGetVariableList</u>	378
<u>axlSetVariable</u>	380
<u>axlUnsetVariable</u>	382

<u>axIShell</u>	383
<u>axIGetAlias</u>	384
<u>axIIsProtectAlias</u>	385
<u>axIProtectAlias</u>	386
<u>axIReadOnlyVariable</u>	387
<u>axISetAlias</u>	389
 10	
User Interface Functions	391
<u>Overview</u>	391
<u>Window Placement</u>	391
<u>Using Menu Files</u>	393
<u>Dynamically Loading Menus</u>	393
<u>Understanding the Menu File Format</u>	394
AXL-SKILL User Interface Functions	398
<u>axICancelOff</u>	398
<u>axICancelOn</u>	399
<u>axICancelTest</u>	401
<u>axIMeterCreate</u>	402
<u>axIMeterDestroy</u>	404
<u>axIMeterUpdate</u>	405
<u>axIMeterIsCancelled</u>	406
<u>axUIMenuLoad</u>	407
<u>axUIMenuDump</u>	408
<u>axUIColorDialog</u>	409
<u>axUIConfirm</u>	410
<u>axUIControl</u>	411
<u>axUIPrompt</u>	413
<u>axIIsViewFileType</u>	415
<u>axUIViewFileCreate</u>	416
<u>axUIViewFileReuse</u>	418
<u>axUIYesNo</u>	420
<u>axUIWExpose</u>	422
<u>axUIWClose</u>	423
<u>axUIWPrint</u>	424

<u>axIUIWRedraw</u>	425
<u>axIUIWBlock</u>	426
<u>axIUIEditFile</u>	427
<u>axIUIMultipleChoice</u>	429
<u>axIUIViewFileScrollTo</u>	430
<u>axIUIWBeep</u>	431
<u>axIUIWDisableQuit</u>	432
<u>axIUIWExposeByName</u>	433
<u>axIUIWPerm</u>	434
<u>axIUIWSetHelpTag</u>	436
<u>axIUIWSetParent</u>	437
<u>axIUIWShow</u>	438
<u>axIUIWTimerAdd</u>	439
<u>axIUIWTimerRemove</u>	441
<u>axIUIWUpdate</u>	442
<u>axIUIYesNoCancel</u>	443
<u>axUIDataBrowse</u>	444

11

<u>Form Interface Functions</u>	447
<u>Overview</u>	447
<u>Programming</u>	447
<u>Field / Control</u>	448
<u>Using Forms Specification Language</u>	457
<u>Moving and Sizing Form Controls During Form Resizing</u>	466
<u>Using Grids</u>	470
<u>AXL-SKILL Form Interface Functions</u>	486
<u>axIFormBNFDoc</u>	486
<u>axIFormCallback</u>	497
<u>axIFormCreate</u>	502
<u>axIFormClose</u>	505
<u>axIFormDisplay</u>	506
<u>axIFormBuildPopup</u>	507
<u>axIFormGetField</u>	511
<u>axIFormListDeleteAll</u>	513

<u>axlFormListSelect</u>	516
<u>axlFormSetField</u>	517
<u>axlFormSetInfo</u>	518
<u>axlFormTest</u>	519
<u>axlFormRestoreField</u>	520
<u>axlFormTitle</u>	522
<u>axlIsFormType</u>	523
<u>axlFormSetFieldVisible</u>	524
<u>axlFormIsFieldVisible</u>	525
<u>Callback Procedure: formCallback</u>	526
<u>axlFormAutoResize</u>	531
<u>axlFormColorize</u>	532
<u>axlFormGetActiveField</u>	535
<u>axlFormGridBatch</u>	536
<u>axlFormGridCancelPopup</u>	537
<u>axlFormGridDeleteRows</u>	538
<u>axlFormGridEvents</u>	539
<u>axlFormGridGetCell</u>	542
<u>axlFormGridInsertCol</u>	544
<u>axlIsGridCellType</u>	548
<u>axlFormGridInsertRows</u>	549
<u>axlFormGridNewCell</u>	550
<u>axlFormGridReset</u>	551
<u>axlFormGridSetBatch</u>	552
<u>axlFormGridUpdate</u>	555
<u>axlFormInvalidateField</u>	556
<u>axlFormIsFieldEditable</u>	557
<u>axlFormListAddItem</u>	558
<u>axlFormListDeleteItem</u>	560
<u>axlFormListGetItem</u>	562
<u>axlFormListGetSelCount</u>	563
<u>axlFormListGetSelItems</u>	564
<u>axlFormListOptions</u>	565
<u>axlFormListSelAll</u>	567
<u>axlFormMsg</u>	568
<u>axlFormGetFieldType</u>	570

<u>axlFormDefaultButton</u>	571
<u>axlFormGridOptions</u>	573
<u>axlFormSetActiveField</u>	575
<u>axlFormSetDecimal</u>	576
<u>axlFormSetFieldEditable</u>	577
<u>axlFormSetFieldLimits</u>	578
<u>axlFormTreeViewAddItem</u>	579
<u>axlFormTreeViewChangeImages</u>	584
<u>axlFormTreeViewChangeLabel</u>	586
<u>axlFormTreeViewGetImages</u>	587
<u>axlFormTreeViewGetLabel</u>	588
<u>axlFormTreeViewGetParents</u>	589
<u>axlFormTreeViewGetSelectState</u>	590
<u>axlFormTreeViewLoadBitmaps</u>	591
<u>axlFormTreeViewSet</u>	593
<u>axlFormTreeViewSetSelectState</u>	595

12

<u>Simple Graphics Drawing Functions</u>	597
<u>Overview</u>	597
<u>Functions</u>	601
<u>axIGRPDrwBitmap</u>	601
<u>axIGRPDrwCircle</u>	602
<u>axIGRPDrwInit</u>	603
<u>axIGRPDrwLine</u>	604
<u>axIGRPDrwMapWindow</u>	605
<u>axIGRPDrwPoly</u>	606
<u>axIGRPDrwRectangle</u>	607
<u>axIGRPDrwText</u>	608
<u>axIGRPDrwUpdate</u>	609

13

Message Handler Functions	611
<u>Overview</u>	611
<u>Message Handler Functions</u>	614
<u>axIMsgPut</u>	614
<u>axIMsgContextPrint</u>	615
<u>axIMsgContextGetString</u>	616
<u>axIMsgContextGet</u>	617
<u>axIMsgContextTest</u>	618
<u>axIMsgContextInBuf</u>	619
<u>axIMsgContextRemove</u>	620
<u>axIMsgContextStart</u>	621
<u>axIMsgContextFinish</u>	622
<u>axIMsgContextClear</u>	623
<u>axIMsgCancelPrint</u>	624
<u>axIMsgCancelSeen</u>	625
<u>axIMsgClear</u>	626
<u>axIMsgSet</u>	627
<u>axIMsgTest</u>	628

14

Design Control Functions	629
<u>AXL-SKILL Design Control Functions</u>	629
<u>axlCurrentDesign</u>	630
<u>axlDesignType</u>	631
<u>axlCompileSymbol</u>	632
<u>axlSetSymbolType</u>	633
<u>axlDBControl</u>	634
<u>axlDBSectorSize</u>	639
<u>axlGetDrawingName</u>	641
<u>axlKillDesign</u>	642
<u>axlOpenDesign</u>	643
<u>axlRenameDesign</u>	644
<u>axlSaveDesign</u>	645

<u>axIDBChangeDesignExtents</u>	646
<u>axIDBChangeDesignOrigin</u>	647
<u>axIDBChangeDesignUnits</u>	648
<u>axIDBCheck</u>	650
<u>axIDBCopyPadstack</u>	652
<u>axIDBDelLock</u>	654
<u>axIDBGetLock</u>	655
<u>axIDBSetLock</u>	656
<u>axIPadStackToDisk</u>	658
<u>axITechnologyType</u>	659
<u>axITriggerClear</u>	660
<u>axITriggerPrint</u>	661
<u>axITriggerSet</u>	662
<u>axIGetActiveLayer</u>	665
<u>axIGetActiveTextBlock</u>	666
<u>axISetActiveLayer</u>	667
15	
Database Group Functions	669
<u>Overview</u>	669
<u>axIDBAddGroupObjects</u>	670
<u>axIDBCreateGroup</u>	671
<u>axIDBDisbandGroup</u>	673
<u>axIDBRemoveGroupObjects</u>	674
16	
Database Attachment Functions	675
<u>Overview</u>	675
<u>axICreateAttachment</u>	676
<u>axIDeleteAttachment</u>	678
<u>axIGetAllAttachmentNames</u>	679
<u>axIGetAttachment</u>	680
<u>axIIsAttachment</u>	682
<u>axISetAttachment</u>	683

17

<u>Database Transaction Functions</u>	685
<u>Overview</u>	685
<u>axIDBCloak</u>	686
<u>axIDBTransactionCommit</u>	688
<u>axIDBTransactionMark</u>	689
<u>axIDBTransactionOops</u>	690
<u>axIDBTransactionRollback</u>	691
<u>axIDBTransactionStart</u>	692

18

<u>Constraint Management Functions</u>	695
<u>Overview</u>	695
<u>axICnsAddVia</u>	696
<u>axICnsAssignPurge</u>	697
<u>axICnsDeleteVia</u>	698
<u>axICNSDesignModeGet</u>	699
<u>axICNSDesignModeSet</u>	701
<u>axICNSDesignValueCheck</u>	704
<u>axICNSDesignValueGet</u>	705
<u>axICNSDesignValueSet</u>	707
<u>axICNSEcsetCreate</u>	709
<u>axICNSEcsetDelete</u>	710
<u>axICNSEcsetGet</u>	711
<u>axICNSEcsetModeGet</u>	712
<u>axICNSEcsetModeSet</u>	714
<u>axICNSEcsetValueCheck</u>	717
<u>axICNSEcsetValueGet</u>	718
<u>axINetEcsetValueGet</u>	721
<u>axICNSEcsetValueSet</u>	723
<u>axICnsGetViaList</u>	725
<u>axIGetAllViaList</u>	726
<u>axILayerSet</u>	727
<u>axICnsList</u>	728

<u>axICNSMapClear</u>	729
<u>axICNSMapUpdate</u>	730
<u>axICnsNetFlattened</u>	732
 19	
<u>Command Control Functions</u>	735
<u>Overview</u>	735
<u>AXL-SKILL Command Control Functions</u>	735
<u>axICmdRegister</u>	736
<u>axICmdUnregister</u>	739
<u>axIEndSkillMode</u>	740
<u>axIFlushDisplay</u>	741
<u>axIOKToProceed</u>	743
<u>axISetLineLock</u>	744
<u>axISetRotateIncrement</u>	746
<u>axIUIGetUserData</u>	747
<u>axIUIPopupDefine</u>	748
<u>axIUIPopupSet</u>	750
<u>axIBuildClassPopup</u>	752
<u>axIBuildSubclassPopup</u>	753
<u>axISubclassFormPopup</u>	755
<u>axIVisibleUpdate</u>	758
<u>axIWindowFit</u>	760
 20	
<u>Polygon Operation Functions</u>	761
<u>Overview</u>	761
<u>About Polygon Operations</u>	761
<u>AXL-SKILL Polygon Operation Attributes</u>	764
<u>AXL-SKILL Polygon Operation Functions</u>	766
<u>axIPolyFromDB</u>	766
<u>axIPolyMemUse</u>	768
<u>axIPolyOffset</u>	770
<u>axIPolyOperation</u>	772
<u>axIPolyExpand</u>	774

<u>axlIsPolyType</u>	780
<u>axlPolyFromHole</u>	781
<u>axlPolyErrorGet</u>	782
<u>Use Models</u>	784
 21	
<u>PCB Editor File Access Functions</u>	787
<u>AXL-SKILL File Access Functions</u>	787
<u>axIDMFileError</u>	788
<u>axIDMFindFile</u>	789
<u>axIDMGetFile</u>	790
<u>axIDMOpenFile</u>	792
<u>axIDMOpenLog</u>	795
<u>axIDMClose</u>	796
<u>axIDMBrowsePath</u>	797
<u>axIDMDirectoryBrowse</u>	798
<u>axIDMFileBrowse</u>	799
<u>axIDMFileParts</u>	801
<u>axIOSFileCopy</u>	802
<u>axIOSFileMove</u>	803
<u>axIOSSlash</u>	804
<u>axlRecursiveDelete</u>	805
<u>axlTempDirectory</u>	807
<u>axlTempFile</u>	808
<u>axlTempFileRemove</u>	809
 22	
<u>Reports and Extract Functions</u>	811
<u>AXL-SKILL Data Extract Functions</u>	811
<u>axlExtractToFile</u>	811
<u>axlExtractMap</u>	813
<u>axlReportList</u>	815
<u>axlReportRegister</u>	816

23

<u>Utility Functions</u>	819
<u>Overview</u>	819
<u>axICmdList</u>	820
<u>axIDebug</u>	821
<u>axIEmail</u>	822
<u>axIGeoArcCenterAngle</u>	824
<u>axIGeoArcCenterRadius</u>	827
<u>axIHttp</u>	832
<u>axILsDebug</u>	834
<u>axILogHeader</u>	835
<u>axIMKS2UU</u>	836
<u>axIMKSAlias</u>	838
<u>axIMKSCConvert</u>	839
<u>axIMKSStr2UU</u>	841
<u>axIMapClassName</u>	842
<u>axIMemSize</u>	844
<u>axIPPrint</u>	845
<u>axIPdfView</u>	846
<u>axIRegexpls</u>	847
<u>axIRunBatchDBProgram</u>	848
<u>axIShowObject</u>	851
<u>axISleep</u>	852
<u>axISort</u>	853
<u>axIStrcmpAlpNum</u>	857
<u>axIVersion</u>	858
<u>axIVersionIdGet</u>	861
<u>axIVersionIdPrint</u>	862

24

<u>Math Utility Functions</u>	863
<u>Overview</u>	863
<u>axIDistance</u>	863
<u>axIGeoEqual</u>	864

<u>axlGeoRotatePt</u>	865
<u>axlIsPointInsideBox</u>	867
<u>axlIsPointOnLine</u>	868
<u>axlLineSlope</u>	869
<u>axlLineXLine</u>	870
<u>axlMPythag</u>	871
<u>axlIMXYAdd</u>	872
<u>axlIMXYSub</u>	873
<u>axl ol ol2</u>	874
 25	
<u>Database Miscellaneous Functions</u>	877
<u>Overview</u>	877
<u>axlAirGap</u>	878
<u>axlExtentDB</u>	881
<u>axlExtentLayout</u>	882
<u>axlExtentSymbol</u>	883
<u>axlDRCGetCount</u>	884
<u>axlGeoPointInShape</u>	885
<u>axlIsHighlighted</u>	886
<u>axlTestPoint</u>	887
<u>axlChangeNet</u>	889
<u>axlSegDelayAndZ0</u>	890
 26	
<u>Logic Access Functions</u>	891
<u>Overview</u>	891
<u>axlIDBCreateConceptComponent</u>	892
<u>axlIDBCreateComponent</u>	894
<u>axlIDBCreateManyModuleInstances</u>	895
<u>axlIDBCreateModuleDef</u>	897
<u>axlIDBCreateModuleInstance</u>	898
<u>axlIDBCreateSymDefSkeleton</u>	900
<u>axlDbidName</u>	901
<u>axlDiffPair</u>	902

<u>axIDiffPairAuto</u>	904
<u>axIDiffPairDBID</u>	906
<u>axIDBGetExtents</u>	907
<u>axIMatchGroupAdd</u>	908
<u>axIMatchGroupCreate</u>	910
<u>axIMatchGroupDelete</u>	912
<u>axIMatchGroupProp</u>	913
<u>axIMatchGroupRemove</u>	915
<u>axIPinPair</u>	916
<u>axIPinPairSeek</u>	919
<u>axIPinsOfNet</u>	920
<u>axIRemoveNet</u>	921
<u>axIRenameNet</u>	922
<u>axIRenameRefdes</u>	924
<u>axIWriteDeviceFile</u>	926
<u>axIWritePackageFile</u>	928

A

<u>Building Contexts in Allegro</u>	929
<u>Introduction</u>	929
<u>Requirements</u>	929
<u>Cautions</u>	929
<u>Building Standard Contexts</u>	930
<u>Building Autoload Contexts</u>	931
<u>Files with This Package</u>	932
<u>File B1</u>	932
<u>File S1</u>	933
<u>File A1</u>	934
<u>File A2</u>	937

Allegro PCB and Package User Guide: SKILL Reference

Alphabetical List of Functions

axl.ol.ol2	874
axlAddSelectAll	221
axlAddSelectBox	219
axlAddSelectName	225
axlAddSelectObject	228
axlAddSelectPoint	214
axlAddSimpleMoveDynamics	341
axlAddSimpleRbandDynamics	339
axlAirGap	878
axlAutoOpenFindFilter	247
axlBuildClassPopup	752
axlBuildSubclassPopup	753
axlCancelEnterFun	345
axlCancelOff	398
axlCancelOn	399
axlCancelTest	401
axlChangeNet	889
axlChangeWidth	260
axlClearDynamics	338
axlClearSelSet	235
axlCloseFindFilter	249
axlCmdList	820
axlCmdRegister	736
axlCmdUnregister	739
axlCnsAddVia	696
axlCnsAssignPurge	697
axlCnsDeleteVia	698
axlCNSDesignModeGet	699
axlCNSDesignModeSet	701
axlCNSDesignValueCheck	704
axlCNSDesignValueGet	705
axlCNSDesignValueSet	707
axlCNSEcsetCreate	709
axlCNSEcsetDelete	710
axlCNSEcsetGet	711
axlCNSEcsetModeGet	712
axlCNSEcsetModeSet	714
axlCNSEcsetValueCheck	717
axlCNSEcsetValueGet	718

<u>axICNSEcsetValueSet</u>	723
<u>axICnsGetViaList</u>	725
<u>axICnsList</u>	728
<u>axICNSMapClear</u>	729
<u>axICNSMapUpdate</u>	730
<u>axICnsNetFlattened</u>	732
<u>axIColorGet</u>	166
<u>axIColorLoad</u>	171
<u>axIColorOnGet</u>	173
<u>axIColorOnSet</u>	175
<u>axIColorPriorityGet</u>	177
<u>axIColorPrioritySet</u>	179
<u>axIColorSave</u>	181
<u>axIColorSet</u>	182
<u>axIColorShadowGet</u>	168
<u>axIColorShadowSet</u>	169
<u>axICreateSymbol</u>	632
<u>axIConductorBottomLayer</u>	200
<u>axIConductorTopLayer</u>	201
<u>axICreateAttachment</u>	360
<u>axICurrentDesign</u>	676
<u>axIDBActiveShape</u>	630
<u>axIDBAddGroupObjects</u>	129
<u>axIDBAddProp</u>	670
<u>axIDBAssignNet</u>	161
<u>axIDBChangeDesignExtents</u>	325
<u>axIDBChangeDesignOrigin</u>	646
<u>axIDBChangeDesignUnits</u>	647
<u>axIDBCheck</u>	648
<u>axIDBCloak</u>	650
<u>axIDBControl</u>	686
<u>axIDBCopyPadstack</u>	634
<u>axIDBCreateCircle</u>	652
<u>axIDBCreateCloseShape</u>	121
<u>axIDBCreateComponent</u>	128
<u>axIDBCreateConceptComponent</u>	894
<u>axIDBCreateExternalDRC</u>	892
<u>axIDBCreateFilmRec</u>	136
<u>axIDBCreateGroup</u>	202
<u>axIDBCreateLine</u>	671
<u>axIDBCreateManyModuleInstances</u>	119
<u>axIDBCreateModuleDef</u>	895
<u>axIDBCreateModuleInstance</u>	897
<u>axIDBCreateModuleInstance</u>	898

Allegro PCB and Package User Guide: SKILL Reference

<u>axIDBCreateOpenShape</u>	125
<u>axIDBCreatePadStack</u>	139
<u>axIDBCreatePath</u>	117
<u>axIDBCreatePin</u>	143
<u>axIDBCreatePropDictEntry</u>	158
<u>axIDBCreateRectangle</u>	134
<u>axIDBCreateShape</u>	132
<u>axIDBCreateSymbol</u>	146
<u>axIDBCreateSymbolAutosilk</u>	157
<u>axIDBCreateSymbolSkeleton</u>	149
<u>axIDBCreateSymDefSkeleton</u>	900
<u>axIDBCreateText</u>	153
<u>axIDBCreateVia</u>	155
<u>axIDBCreateVoid</u>	131
<u>axIDBCreateVoidCircle</u>	130
<u>axIDBDeleteProp</u>	264
<u>axIDBDelLock</u>	654
<u>axIDBDisbandGroup</u>	673
<u>axIDBDynamicShapes</u>	326
<u>axIDBGetAttachedText</u>	297
<u>axIDBGetConnect</u>	308
<u>axIDBGetDesign</u>	294
<u>axIDBGetDesignUnits</u>	304
<u>axIDBGetDrillPlating</u>	295
<u>axIDBGetExtents</u>	907
<u>axIDBGetLayerType</u>	187
<u>axIDBGetLock</u>	655
<u>axIDBGetLonelyBranches</u>	307
<u>axIDBGetManhattan</u>	309
<u>axIDBGetPad</u>	299
<u>axIDBGetPropDictEntry</u>	301
<u>axIDBGetProperties</u>	302
<u>axIDBGetShapes</u>	327
<u>axIDBIdName</u>	901
<u>axIDBIsBondingWireLayer</u>	329
<u>axIDBIsBondpad</u>	310
<u>axIDBIsBondwire</u>	311
<u>axIDBIsDiePad</u>	312
<u>axIDBIsFixed</u>	313
<u>axIDBIsPackagePin</u>	314
<u>axIDBIsPlatingbarPin</u>	315
<u>axIDBRefreshId</u>	305
<u>axIDBRemoveGroupObjects</u>	674
<u>axIDBSectorSize</u>	639

<u>axIDBSetLock</u>	656
<u>axIDBTextBlockCompact</u>	330
<u>axIDBTransactionCommit</u>	688
<u>axIDBTransactionMark</u>	689
<u>axIDBTransactionOps</u>	690
<u>axIDBTransactionRollback</u>	691
<u>axIDBTransactionStart</u>	692
<u>axIDebug</u>	821
<u>axIDehighlightObject</u>	353
<u>axIDeleteAttachment</u>	678
<u>axIDeleteFillet</u>	271
<u>axIDeleteObject</u>	262
<u>axIDesignType</u>	631
<u>axIDiffPair</u>	902
<u>axIDiffPairAuto</u>	904
<u>axIDiffPairDBID</u>	906
<u>axIDistance</u>	863
<u>axIDMBrowsePath</u>	797
<u>axIDMClose</u>	796
<u>axIDMDirectoryBrowse</u>	798
<u>axIDMFileBrowse</u>	799
<u>axIDMFileError</u>	788
<u>axIDMFileParts</u>	801
<u>axIDMFultipartFile</u>	789
<u>axIDMGetFile</u>	790
<u>axIDMOpenFile</u>	792
<u>axIDMOpenLog</u>	795
<u>axIDrawObject</u>	357
<u>axIDRCGetCount</u>	884
<u>axIDynamicsObject</u>	358
<u>axIEmail</u>	822
<u>axEndSkillMode</u>	740
<u>axEnterAngle</u>	344
<u>axEnterBox</u>	348
<u>axEnterEvent</u>	361
<u>axEnterPath</u>	350
<u>axEnterPoint</u>	342
<u>axEnterString</u>	343
<u>axEraseObject</u>	359
<u>axEventSetStartPopup</u>	365
<u>axExtentDB</u>	881
<u>axExtentLayout</u>	882
<u>axExtentSymbol</u>	883
<u>axExtractMap</u>	813

<u>axlExtractToFile</u>	811
<u>axlFillet</u>	272
<u>axlFindFilterIsOpen</u>	250
<u>axlFinishEnterFun</u>	346
<u>axlFlushDisplay</u>	741
<u>axlFormAutoResize</u>	531
<u>axlFormBNFDoc</u>	486
<u>axlFormBuildPopup</u>	507
<u>axlFormCallback</u>	497
<u>axlFormClose</u>	505
<u>axlFormColorize</u>	532
<u>axlFormCreate</u>	502
<u>axlFormDefaultButton</u>	571
<u>axlFormDisplay</u>	506
<u>axlFormGetActiveField</u>	535
<u>axlFormGetField</u>	511
<u>axlFormGetFieldType</u>	570
<u>axlFormGridBatch</u>	536
<u>axlFormGridCancelPopup</u>	537
<u>axlFormGridDeleteRows</u>	538
<u>axlFormGridEvents</u>	539
<u>axlFormGridGetCell</u>	542
<u>axlFormGridInsertCol</u>	544
<u>axlFormGridInsertRows</u>	549
<u>axlFormGridNewCell</u>	550
<u>axlFormGridOptions</u>	573
<u>axlFormGridReset</u>	551
<u>axlFormGridSetBatch</u>	552
<u>axlFormGridUpdate</u>	555
<u>axlFormInvalidateField</u>	556
<u>axlFormIsFieldEditable</u>	557
<u>axlFormIsFieldVisible</u>	525
<u>axlFormListAddItem</u>	558
<u>axlFormListDeleteAll</u>	513
<u>axlFormListDeleteItem</u>	560
<u>axlFormListGetItem</u>	562
<u>axlFormListGetSelCount</u>	563
<u>axlFormListGetSelItems</u>	564
<u>axlFormListOptions</u>	565
<u>axlFormListSelAll</u>	567
<u>axlFormListSelect</u>	516
<u>axlFormMsg</u>	568
<u>axlFormRestoreField</u>	520
<u>axlFormSetActiveField</u>	575

<u>axlFormSetDecimal</u>	576
<u>axlFormSetField</u>	517
<u>axlFormSetFieldEditable</u>	577
<u>axlFormSetFieldLimits</u>	578
<u>axlFormSetFieldVisible</u>	524
<u>axlFormSetInfo</u>	518
<u>axlFormTest</u>	519
<u>axlFormTitle</u>	522
<u>axlFormTreeViewAddItem</u>	579
<u>axlFormTreeViewChangeImages</u>	584
<u>axlFormTreeViewChangeLabel</u>	586
<u>axlFormTreeViewGetImages</u>	587
<u>axlFormTreeViewGetLabel</u>	588
<u>axlFormTreeViewGetParents</u>	589
<u>axlFormTreeViewGetSelectState</u>	590
<u>axlFormTreeViewLoadBitmaps</u>	591
<u>axlFormTreeViewSet</u>	593
<u>axlFormTreeViewSetSelectState</u>	595
<u>axlGeoArcCenterAngle</u>	824
<u>axlGeoArcCenterRadius</u>	827
<u>axlGeoEqual</u>	864
<u>axlGeoPointInShape</u>	885
<u>axlGeoRotatePt</u>	865
<u>axlGetActiveLayer</u>	665
<u>axlGetActiveTextBlock</u>	666
<u>axlGetAlias</u>	384
<u>axlGetAllAttachmentNames</u>	679
<u>axlGetAllViaList</u>	726
<u>axlGetAttachment</u>	680
<u>axlGetDrawingName</u>	641
<u>axlGetDynamicsSegs</u>	347
<u>axlGetFindFilter</u>	236
<u>axlGetLastEnterPoint</u>	267
<u>axlGetModuleInstanceDefinition</u>	316
<u>axlGetModuleInstanceLocation</u>	317
<u>axlGetModuleInstanceLogicMethod</u>	318
<u>axlGetModuleInstanceNetExceptions</u>	319
<u>axlGetParam</u>	185
<u>axlGetSelSet</u>	232
<u>axlGetSelSetCount</u>	234
<u>axlGetTrapBox</u>	367
<u>axlGetVariable</u>	376
<u>axlGetVariableList</u>	378
<u>axlGRPDrwBitmap</u>	601

<u>axIGRPDrwCircle</u>	602
<u>axIGRPDrwInit</u>	603
<u>axIGRPDrwLine</u>	604
<u>axIGRPDrwMapWindow</u>	605
<u>axIGRPDrwPoly</u>	606
<u>axIGRPDrwRectangle</u>	607
<u>axIGRPDrwText</u>	608
<u>axIGRPDrwUpdate</u>	609
<u>axHighlightObject</u>	351
<u>axHttp</u>	832
<u>axIIsAttachment</u>	682
<u>axIIsDBIDType</u>	296
<u>axIIsDebug</u>	834
<u>axIIsDummyNet</u>	320
<u>axIIsFormType</u>	523
<u>axIIsGridCellType</u>	548
<u>axIIsHighlighted</u>	886
<u>axIIsitFill</u>	323
<u>axIIsLayer</u>	188
<u>axIIsLayerNegative</u>	321
<u>axIIsPinUnused</u>	322
<u>axIIsPointInsideBox</u>	867
<u>axIIsPointOnLine</u>	868
<u>axIIsPolyType</u>	780
<u>axIIsProtectAlias</u>	385
<u>axIIsViewFileType</u>	415
<u>axIIsVisibleLayer</u>	189
<u>axKillDesign</u>	642
<u>axILayerCreateCrossSection</u>	190
<u>axILayerCreateNonConductor</u>	192
<u>axILayerGet</u>	193
<u>axILayerSet</u>	727
<u>axILineSlope</u>	869
<u>axILineXLine</u>	870
<u>axILoadPadstack</u>	164
<u>axILogHeader</u>	835
<u>axIMakeDynamicsPath</u>	373
<u>axIMapClassName</u>	842
<u>axIMatchGroupAdd</u>	908
<u>axIMatchGroupCreate</u>	910
<u>axIMatchGroupDelete</u>	912
<u>axIMatchGroupProp</u>	913
<u>axIMatchGroupRemove</u>	915
<u>axIMemSize</u>	844

<u>axIMeterCreate</u>	402
<u>axIMeterDestroy</u>	404
<u>axIMeterIsCancelled</u>	406
<u>axIMeterUpdate</u>	405
<u>axIMiniStatusLoad</u>	354
<u>axIMKS2UU</u>	836
<u>axIMKSAlias</u>	838
<u>axIMKSCConvert</u>	839
<u>axIMKSStr2UU</u>	841
<u>axIMPythag</u>	871
<u>axIMsgCancelPrint</u>	624
<u>axIMsgCancelSeen</u>	625
<u>axIMsgClear</u>	626
<u>axIMsgContextClear</u>	623
<u>axIMsgContextFinish</u>	622
<u>axIMsgContextGet</u>	617
<u>axIMsgContextGetString</u>	616
<u>axIMsgContextInBuf</u>	619
<u>axIMsgContextPrint</u>	615
<u>axIMsgContextRemove</u>	620
<u>axIMsgContextStart</u>	621
<u>axIMsgContextTest</u>	618
<u>axIMsgPut</u>	614
<u>axIMsgSet</u>	627
<u>axIMsgTest</u>	628
<u>axIMXYAdd</u>	872
<u>axIMXYSub</u>	873
<u>axINetEcsetValueGet</u>	721
<u>axIOK2Void</u>	324
<u>axIOKToProceed</u>	743
<u>axOpenDesign</u>	643
<u>axOpenFindFilter</u>	248
<u>axIOSFileCopy</u>	802
<u>axIOSFileMove</u>	803
<u>axOSSlash</u>	804
<u>axPadStackToDisk</u>	658
<u>axPathArcAngle</u>	104
<u>axPathArcCenter</u>	104
<u>axPathArcRadius</u>	104
<u>axPathGetLastPathSeg</u>	112
<u>axPathGetPathSegs</u>	111
<u>axPathGetWidth</u>	109
<u>axPathLine</u>	108
<u>axPathSegGetArcCenter</u>	114

Allegro PCB and Package User Guide: SKILL Reference

<u>axIPathSegGetArcClockwise</u>	115
<u>axIPathSegGetEndPoint</u>	113
<u>axIPathSegGetWidth</u>	110
<u>axIPathStart</u>	102
<u>axIPathStartCircle</u>	116
<u>axIPdfView</u>	846
<u>axIPinPair</u>	916
<u>axIPinPairSeek</u>	919
<u>axIPinsOfNet</u>	920
<u>axIPolyErrorGet</u>	782
<u>axIPolyExpand</u>	774
<u>axIPolyFromDB</u>	766
<u>axIPolyFromHole</u>	781
<u>axIPolyMemUse</u>	768
<u>axIPolyOffset</u>	770
<u>axIPolyOperation</u>	772
<u>axIPPrint</u>	845
<u>axIProtectAlias</u>	386
<u>axIPurgePadstacks</u>	273
<u>axIRatsnestBlank</u>	368
<u>axIRatsnestDisplay</u>	369
<u>axIReadOnlyVariable</u>	387
<u>axIRecursiveDelete</u>	805
<u>axIREgexps</u>	847
<u>axIRemoveNet</u>	921
<u>axIRenameDesign</u>	644
<u>axIRenameNet</u>	922
<u>axIRenameRefdes</u>	924
<u>axIRotatePadstack</u>	270
<u>axIReportList</u>	815
<u>axIReportRegister</u>	816
<u>axIRunBatchDBProgram</u>	848
<u>axISaveDesign</u>	645
<u>axISegDelayAndZ0</u>	890
<u>axISelect</u>	230
<u>axISelectByName</u>	251
<u>axISelectByProperty</u>	256
<u>axISetActiveLayer</u>	667
<u>axISetAlias</u>	389
<u>axISetAttachment</u>	683
<u>axISetFindFilter</u>	237
<u>axISetLineLock</u>	744
<u>axISetParam</u>	186
<u>axISetPlaneType</u>	205

<u>axlSetRotateIncrement</u>	746
<u>axlSetSymbolType</u>	633
<u>axlSetVariable</u>	380
<u>axlShapeAutoVoid</u>	275
<u>axlShapeChangeDynamicType</u>	277
<u>axlShapeDeleteVoids</u>	279
<u>axlShapeDynamicUpdate</u>	281
<u>axlShapeRaisePriority</u>	282
<u>axlShell</u>	383
<u>axlShovelItems</u>	284
<u>axlShoveSetParams</u>	285
<u>axlShowObject</u>	851
<u>axlShowObjectToFile</u>	370
<u>axlSingleSelectBox</u>	217
<u>axlSingleSelectName</u>	223
<u>axlSingleSelectObject</u>	227
<u>axlSingleSelectPoint</u>	212
<u>axlSleep</u>	852
<u>axlSort</u>	853
<u>axlStrcmpAlpNum</u>	857
<u>axlSubclassFormPopup</u>	755
<u>axlSubSelectAll</u>	222
<u>axlSubSelectBox</u>	220
<u>axlSubSelectName</u>	226
<u>axlSubSelectObject</u>	229
<u>axlSubSelectPoint</u>	215
<u>axlTechnologyType</u>	659
<u>axlTempDirectory</u>	807
<u>axlTempFile</u>	808
<u>axlTempFileRemove</u>	809
<u>axlTestPoint</u>	887
<u>axlTransformObject</u>	288
<u>axlTriggerClear</u>	660
<u>axlTriggerPrint</u>	661
<u>axlTriggerSet</u>	662
<u>axlUICmdPopupSet</u>	371
<u>axlUIColorDialog</u>	409
<u>axlUIConfirm</u>	410
<u>axlUIControl</u>	411
<u>axlUIDataBrowse</u>	444
<u>axlUIEditFile</u>	427
<u>axlUIGetUserData</u>	747
<u>axlUIMenuDump</u>	408
<u>axlUIMenuLoad</u>	407

Allegro PCB and Package User Guide: SKILL Reference

<u>axlUIMultipleChoice</u>	429
<u>axlUIPopupDefine</u>	748
<u>axlUIPopupSet</u>	750
<u>axlUIPrompt</u>	413
<u>axlUIViewFileCreate</u>	416
<u>axlUIViewFileReuse</u>	418
<u>axlUIViewFileScrollTo</u>	430
<u>axlUIWBeep</u>	431
<u>axlUIWBlock</u>	426
<u>axlUIWClose</u>	423
<u>axlUIWDisableQuit</u>	432
<u>axlUIWExpose</u>	422
<u>axlUIWExposeByName</u>	433
<u>axlUIWPerm</u>	434
<u>axlUIWPrint</u>	424
<u>axlUIWRedraw</u>	425
<u>axlUIWSetHelpTag</u>	436
<u>axlUIWSetParent</u>	437
<u>axlUIWShow</u>	438
<u>axlUIWTimerAdd</u>	439
<u>axlUIWTimerRemove</u>	441
<u>axlUIWUpdate</u>	442
<u>axlUIYesNo</u>	420
<u>axlUIYesNoCancel</u>	443
<u>axlUnsetVariable</u>	382
<u>axlVersion</u>	858
<u>axlVersionIdGet</u>	861
<u>axlVersionIdPrint</u>	862
<u>axlVisibleDesign</u>	194
<u>axlVisibleGet</u>	196
<u>axlVisibleLayer</u>	198
<u>axlVisibleSet</u>	199
<u>axlVisibleUpdate</u>	758
<u>axlWindowBoxGet</u>	268
<u>axlWindowBoxSet</u>	269
<u>axlWindowFit</u>	760
<u>axlWriteDeviceFile</u>	926
<u>axlWritePackageFile</u>	928
<u>axlZoomToDbid</u>	372
<u>Callback Procedure: formCallback</u>	526
<u>Cautions</u>	929
<u>Field / Control</u>	448
<u>File A1</u>	934
<u>File A2</u>	937

Allegro PCB and Package User Guide: SKILL Reference

<u>File B1</u>	932
<u>File S1</u>	933
<u>Programming</u>	447
<u>Requirements</u>	929

Before You Start

About This Manual

This manual is for designers and engineers who use the PCB Editor SKILL functions to customize existing PCB Editor interactive commands or create new ones. It describes the AXL (Allegro eXtension Language) user model, how to start AXL, and how to use each AXL function.

This manual assumes that you are familiar with the development and design of printed circuit boards (PCBs). It also assumes you are familiar with PCB Editor for the physical design of PCBs and analysis of reliability, testability, and manufacturability.

If you are reading this manual for a general understanding of AXL capabilities, but do not actually intend to program in AXL, read [Chapter 1, “Introduction to PCB Editor SKILL Functions.”](#)

You should also be familiar with the Cadence SKILL language. The following manuals describe SKILL:

- *SKILL Language User Guide*
- *SKILL Language Reference*
- *Cadence SKILL Functions Quick Reference*

Prerequisites

Before you begin using PCB Editor to design boards, you should be familiar with the PCB Editor environment.

The *Allegro PCB and Package User Guide: Getting Started with Physical Design* explains how to:

- Start PCB Editor
- Navigate in the PCB Editor software
- Get help on a command
- Use the mouse, menus and forms
- Start a design session

Command Syntax Conventions

AXL-SKILL descriptions adhere to the conventions described in the *SKILL Language Basics*. In addition, this manual uses the conventions described below.

<i>italics</i>	Data type name.
<i>nil</i>	Standard SKILL for empty list; as a return value, may indicate failure.
<i>t</i>	Standard SKILL for “true;” return value for success.
<i>[name]</i>	Optional argument “name.”
<i><name></i>	Argument of type “name” required.
<i>dbid</i>	Instance of an PCB Editor database object (means “database id”)
<i>figure</i>	Geometric PCB Editor database object—for example, line, shape, or symbol are all PCB Editor figures. This is not to be confused with the PCB Editor’s database object “figure”, a special graphic denoting DRC markers and drill holes. To ensure clear distinction in this manual, “PCB Editor figure” denotes this special object.

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

<i>l_bBox</i>	A list of the coordinates of a bounding box. Coordinate pairs are lower left and upper right. For example,
	<code>(list(100:100 200:200))</code>
<i>t_layer</i>	A pair of names separated by a slash “/” denoting the name of an PCB Editor class-subclass. For example, “PACKAGE GEOMETRY/SILKSCREEN_TOP”
<i>lo_dbid</i>	Function that takes or returns a dbid or list of dbids.
<i>o_dbid</i>	Function that takes or returns a single dbid.

Referencing Objects by Name

When programming AXL, you can select or refer to a named object by using the unique name of that object. The table shows PCB Editor object types and their associated names:

Table P-1 PCB Editor Object Types and Names

PCB Editor Object Type	Name
NET	netname
COMPONENT	refdes
SYMBOL	refdes or symbol pin: <refdes>.<pin number>
FUNCTION	function designator
DEVTYPE	device type
SYMTYPE	symbol type, for example, “DIP 14”
PROPERTY	property name, for example, “MAX_OVERSHOOT”

You can select objects using the `axlName` functions in [Chapter 5, “Selection and Find Functions”](#).

Finding Information in This Manual

The following table summarizes the topics described in this manual.

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
AXL operation and the relation between PCB Editor database and AXL functions:	<u>Chapter 1, “Introduction to PCB Editor SKILL Functions”</u>
<ul style="list-style-type: none">■ AXL functions■ AXL initialization, environment■ Starting and stopping AXL■ Debugging AXL programs■ <i>dbids</i> and object persistence■ Selecting PCB Editor database objects■ AXL–SKILL database object types	
The structure of PCB Editor AXL database objects, and how they are related to each other. Lists attributes of each object type.	<u>Chapter 2, “The PCB Editor Database User Model”</u>
<ul style="list-style-type: none">■ Database rules■ Attribute types■ Figure (geometric) database types■ Logical database types■ Property database types■ Full attribute listing for all PCB Editor database objects	
Path structures and AXL functions that add path objects and other figure objects to the PCB Editor database.	<u>Chapter 3, “Database Create Functions”</u>
<ul style="list-style-type: none">■ Path create functions■ Create shape and rectangle functions■ Create functions for line, pin, symbol, text, and via	

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
Read/Write access to PCB Editor database parameter objects.	<u>Chapter 4, “Parameter Management Functions”</u>
The select set and AXL functions for managing the select set and selecting single and multiple database objects.	<u>Chapter 5, “Selection and Find Functions”</u>
<ul style="list-style-type: none">■ Point selection■ Box selection■ Selection by name and object <i>dbid</i>■ Find filter management■ Selection set management	
AXL functions that operate on database objects in the same way as interactive PCB Editor commands, including functions to:	<u>Chapter 6, “Interactive Edit Functions”</u>
<ul style="list-style-type: none">■ Delete objects■ Show objects	
Reading database objects:	<u>Chapter 7, “Database Read Functions”</u>
<ul style="list-style-type: none">■ Opening an PCB Editor design■ Accessing standalone branch figures, properties, pads, and text	
AXL functions for	<u>Chapter 8, “PCB Editor Interface Functions”</u>
<ul style="list-style-type: none">■ Highlighting and displaying database objects■ Loading the cursor buffer and dynamic rubberband displays for interactive commands■ Accepting single and multiple user coordinate picks■ Callback functions for completing and cancelling commands	

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
AXL functions for	<u>Chapter 9, “PCB Editor Command Shell Functions”</u>
■ Setting PCB Editor shell variables	
■ Sending a command string to the PCB Editor shell.	
AXL functions for	<u>Chapter 10, “User Interface Functions”</u>
■ Prompting and getting confirmation from the user, displaying text files	
■ Displaying and printing ASCII files	
AXL forms and functions for	<u>Chapter 11, “Form Interface Functions”</u>
■ Creating forms, including the various types of form fields	
■ Setting up callbacks for response to input to individual fields	
AXL functions related to Simple Graphics Drawing	<u>Chapter 12, “Simple Graphics Drawing Functions”</u>
Writing AXL functions for	<u>Chapter 13, “Message Handler Functions”</u>
■ Setting up for user messages	
■ Displaying messages to users	

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
AXL functions for	<u>Chapter 14, “Design Control Functions”</u>

- Opening a design
- Compiling, running edit check, and saving the current (symbol) design
- Getting the type of the active design
- Setting the PCB Editor symbol type
- Getting or setting the value for a specified database control, sector, and obstacle
- Changing the extents, origin, units and accuracy of the design
- Setting, deleting, and getting information about locks on the database
- Setting, clearing, and printing information about triggers, which register interest in events that occur in PCB Editor
- Getting the active class and subclass, active text block, type of design technology in use, and the full path of the drawing
- Saving the design
- Saving a board padstack out to a library
- Creating a new padstack by copying from an existing padstack
- Running dbdoctor on the current database

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
AXL functions for	<u>Chapter 15, “Database Group Functions”</u>
<ul style="list-style-type: none">■ Creating and removing database groups.■ Adding and removing database objects from database groups.	
AXL functions for	<u>Chapter 16, “Database Attachment Functions”</u>
<ul style="list-style-type: none">■ Creating, changing, and deleting database attachments■ Checking whether an object is a database attachment■ Getting the ids of all database attachments■ Getting a database attachment	
AXL functions for	<u>Chapter 17, “Database Transaction Functions”</u>
<ul style="list-style-type: none">■ Improving the performance and program memory use while updating many etch or package symbols in batch mode■ Marking the start of a database transaction and returning the mark to the calling function■ Writing a mark in the database to allow future rollback or commitment■ Committing and undoing a database transaction	

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
AXL functions for	
<ul style="list-style-type: none">■ Getting and setting current DRC modes and values for design constraints and ECset members■ Creating, deleting, and getting the dbid of an ECset■ Checking the syntax of a given value against the allowed syntax for a given constraint■ Batching and tuning DRC updates from constraint changes made using axlCNS<xxx> functions	<u>Chapter 18, “Constraint Management Functions”</u>
AXL functions for	<u>Chapter 19, “Command Control Functions”</u>
<ul style="list-style-type: none">■ Registering and unregistering SKILL commands with the command interpreter■ Getting and setting the controls for line lock, active layer■ Defining popups■ Getting user data	
Polygon operation functions, attributes, and use models.	<u>Chapter 20, “Polygon Operation Functions”</u>
Getting PCB Editor file names, and opening and closing files	<u>Chapter 21, “PCB Editor File Access Functions”</u>
AXL functions for	<u>Chapter 22, “AXL-SKILL Data Extract Functions”</u>
<ul style="list-style-type: none">■ SKILL access to the extract command■ Selecting sets of database objects as members of a view and applying an AXL function to each	

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About . . .	Read . . .
AXL functions for	<u>Chapter 23, “Utility Functions”</u>
■ Calculating an arc center given various data	
■ Converting quantities to various units	
Using math utility functions.	<u>Chapter 24, “Math Utility Functions”</u>
Odds and ends.	<u>Chapter 25, “Database Miscellaneous Functions”</u>
Using logic access functions.	<u>Chapter 26, “Logic Access Functions”</u>

Other Sources of Information

For more information about PCB Editor and other related products, consult the sources listed below.

Product Installation

The *Cadence Installation Guide* tells you how to install Cadence products.

Related Manuals

The following manuals comprise the PCB Editor documentation set for your workbench of PCB design tools:

For Information About...	Read...
The PCB Editor user interface. An overview of the design process using PCB Editor, starting and exiting, controlling the graphic display, graphic and text elements, design information, and system information.	<u>Allegro PCB and Package User Guide: Getting Started with Physical Design</u>
Building and managing libraries, including defining padstacks, custom pads, packages, electrical attributes, and formats.	<u>Allegro PCB and Package User Guide: Defining and Developing Libraries</u>

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About...	Read...
Loading logical design data and converting third-party mechanical data, including loading data from Concept™, netlists, and board mechanical data.	<u>Allegro PCB and Package User Guide: Transferring Logic Design Data</u>
Setting up the design and specifying design rules and controls, including instructions for specifying properties and constraints.	<u>Allegro PCB and Package User Guide: Preparing the Layout</u>
Placing components using PCB Editor, including automatic and interactive placement.	<u>Allegro PCB and Package User Guide: Placing the Elements</u>
Routing using PCB Editor, including interactive and automatic routing.	<u>Allegro PCB and Package User Guide: Routing the Design</u>
Design output, including renaming reference designators, creating drill and silkscreen data, and generating penplots.	<u>Allegro PCB and Package User Guide: Preparing Manufacturing Data</u>
Optional PCB Editor interfaces, including Cadnetix-E, CBDS, Racal Visula, IGES, Greenfield, Computervision CADDs, SDRC I-DEAS, AutoCAD DXF, PTC, CATIA, IPC-D_350C, GDSII, Fluke Defect Analyzer, and HP3070 Tester.	<u>Converting Third Party Designs and Mechanical Data</u>
PCB Editor commands, listed alphabetically.	<u>Allegro PCB and Package Physical Layout Command Reference</u>
PCB Editor properties, extracts (examples), and reports.	<u>Allegro PCB and Package User Guide: Design Rules, Extract Data Dictionary, and Viewing Reports On Screen in HTML Format</u>

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

For Information About...

A comprehensive glossary for the PCB Editor user guides and reference manuals.

Read...

[*Allegro PCB and
Package User Guide
Glossary*](#)

Customer Support

Cadence offers many customer education services. Ask your sales representative for more information.

Customer support is available for customers who have a maintenance agreement with Cadence. Contact Cadence Customer Support at <http://sourcelink.cadence.com>

SourceLink

You can also find technical information, including SKILL documentation and shareware code, online through SourceLink at:

<http://sourcelink.cadence.com>

SourceLink provides the latest in quarterly software rollups (QSRs), case and product change release (PCR) information, technical documentation, solutions, software updates and more.

PCB Editor Users Mailing List Subscription

You can use electronic mail (email) to subscribe to the PCB Editor users mailing list. You will receive periodic newsletters containing up-to-date information on the PCB Editor product family.

To subscribe to the PCB Editor mailing list

- Send an email message to *majordomo@cadence.com*, and include the phrase "subscribe allegro_users" in the body of the message.

You will receive an acceptance notification.

AXL-SKILL Example Files

You can find AXL-SKILL example files in this location:

`%cds_inst_dir%/share/pcb/examples/skill`

User Discussion Forums

You can find discussion groups for users of Cadence products at:

<http://www.cadenceusers.org>

Allegro PCB and Package User Guide: SKILL Reference

Before You Start

Introduction to PCB Editor SKILL Functions

Overview

This chapter is a brief overview of the following:

- PCB Editor AXL-SKILL
- AXL-SKILL functions
- How to initialize and run AXL-SKILL
- The AXL PCB Editor database

Later chapters describe in detail the AXL database objects and all AXL-SKILL functions.

AXL-SKILL in PCB Editor

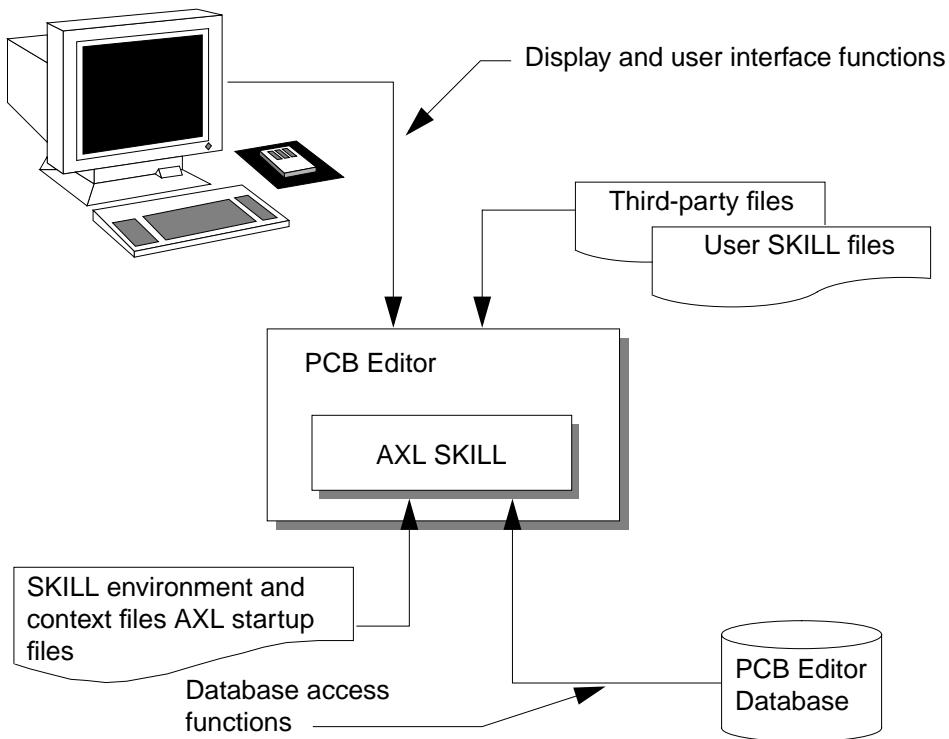
AXL-SKILL is a language processor contained in PCB Editor, as shown in the following figure.

AXL-SKILL contains and is an extension of the core Cadence SKILL language. You use AXL-SKILL functions to access the PCB Editor database and its display and user interfaces. Once you have accessed the PCB Editor database, you can process the data using the core

Allegro PCB and Package User Guide: SKILL Reference

Introduction to PCB Editor SKILL Functions

SKILL functions. The *SKILL Language Functions Reference* describes core SKILL and its available functions.



You access AXL-SKILL by entering the command `skill` on the PCB Editor command line.

AXL-SKILL initializes automatically when you start PCB Editor. As PCB Editor starts, it reads the PCB Editor `env` file, then the AXL `ilinit` file (as described below) then any script you may have specified with the `-s` option in the UNIX command starting PCB Editor.

PCB Editor looks for the `ilinit` file in the following way:

- For the locations specified below, PCB Editor reads one of the following files:

```
<program_name>.ilinit  
allegro.ilinit
```

From the locations:

```
<cdsroot>/share pcb/etc/skill  
<cdssite>/pcb/skill
```

Note: `<cdssite>` defaults to `<cdsroot>/share/local/pcb/skill`, otherwise you can set the `CDS_SITE` variable to point to another location, the directory where your

program started:

```
$HOME/pcbenv
```

If you have multiple `iilinit` files in the locations listed above, *each* of the `iilinit` files will be read. If you wish only the *first* found `iilinit` file to be read (the methodology employed in pre-14.2 releases), set the environment variable `skill_old_iilinit`.

Note: You cannot insert SKILL commands in the `env` file.

Running AXL-SKILL from the Command Line

You run AXL-SKILL by typing `skill` on the PCB Editor command line. The AXL-SKILL interpreter appears with the `skill>` prompt in place of the PCB Editor command line.

You can also run AXL-SKILL functions from the PCB Editor command line with the following syntax:

```
skill (<function> <arguments>)
```

Type `exit` to close the AXL-SKILL interpreter and return to the PCB Editor command line.

Running AXL-SKILL in Batch Mode

You can run AXL-SKILL in batch mode using an X terminal window without displaying PCB Editor, by typing the following command:

```
.allegro -nographic
```

If your system is not running the X window system, verify that it has its `DISPLAY` environment variable set to a system running an X-server.

Note: The `-nographic` switch is valid for all PCB Editor graphic executables, such as `allegro_layout`, `allegro_engineer`, `allegro_prep`, `allegro_interactive`, and `allegro_layout`.

You can also program the PCB Editor and AXL-SKILL startup and command entry in a shell script, allowing full batch capability.

Debugging AXL-SKILL Programs

If you have a SKILL ACCESS development license, you can debug AXL-SKILL programs in PCB Editor using the same tools offered by other Cadence SKILL programs. See *SKILL Language Functions Reference* for a description of those tools, which are primarily available on Unix.

AXL-SKILL Grammar

AXL-SKILL functions follow SKILL grammar rules. In addition, the following characteristics apply to most AXL-SKILL functions:

- All PCB Editor AXL function names begin with axl.
- AXL functions are classified into families that typically have similar calling sequences and share a common part of their names, for example the `axlDBCreate` family, with members such as `axlDBCreateShape` and `axlDBCreateSymbol`.

SKILL Development Window

You can get a larger SKILL-only window by setting the PCB Editor environment variable, `TELSKILL`.

Note: All PCB Editor console output is directed to the SKILL window when the `TELSKILL` variable is set.

AXL-SKILL Database

PCB Editor stores design data as various types of objects in a proprietary database format. These object types can create a complete representation of an electronic layout. You can create, operate on, and extract information from this database using AXL-SKILL programs.

The AXL-SKILL database stores both physical and logical information about your design. Physical information is objects such as geometrical shapes (connecting etch, for example). Logical information is objects such as nets and logical components.

Object dbids (database identifiers)

Every PCB Editor database object has a unique *dbid* (database identifier) associated with it. When you call a function to operate on a database object, you identify the object to the function by giving the object's *dbid* as an argument.

Only AXL routines can create *dbids*. You cannot alter *dbids* directly, except for parameter record ids.

Out of Scope dbids

When a *dbid* is separated from its object, the *dbid* is considered out-of-scope. A *dbid* can become separated from its object for any of the following reasons:

- You delete an object.
- You add a connect line that touches an existing connect line. This causes the existing line to break in to two objects, each with a separate dbid, so the original dbid of the line no longer denotes the same object.
- You return to PCB Editor from AXL.
- You run an PCB Editor command while in SKILL, using the AXL shell function. To minimize out-of-scope issues with AXL shell, isolate AXL shell functions and if necessary refresh dbids after AXL shell calls with the function axlDBRefreshID.
- You open a new layout.

Out of scope *dbids* have no attributes, so they have no object type. If you try to evaluate a *dbid* that is out of scope, SKILL displays the message:

```
dbid: removed
```

Using an out of scope *dbid* in a function causes unexpected results.

Object Types

Each PCB Editor object has an associated *type* and a set of *attributes* that describe the object. For example, all `symbol` objects are of type `symbol`. All `symbols` have the `isMirrored` attribute, among others, and this attribute has the value `t` if the symbol is mirrored or `nil` if it is not. You can use an AXL-SKILL function with "`->`" (the access operator) to access any object attribute. If the attribute does not exist for that object the function returns `nil`.

You use the SKILL special attributes `?` (question mark) and `??` to see all attributes and all attribute/value pairs of an object. See [Chapter 2, “The PCB Editor Database User Model,”](#) for more information about object types.

Object Classes

An *object class* is a data-type abstraction used to group related object types. When a number of distinct object types share enough attributes, you can discuss them as a single class. The start of each section describing a class of object types lists all attributes common to that class.

The different types and classes of objects form a class hierarchy. At the top of the hierarchy is a class containing all types. At the bottom of the hierarchy, each leaf (terminal node) represents an object type that you can create, delete, and save on disk. Each intermediate node in the hierarchy has attributes that are common to all of its children. AXL-SKILL figures,

for example, have common attributes of `layer` and `bBox` (bounding box). These higher level classes do not exist as objects.

Select Sets and Find Functions

AXL-SKILL edit functions obtain the identities of the objects on which they operate from a list of `dbids` called the *select set*. You accumulate `dbids` in the select set by selecting one or several objects using AXL select functions. You then apply edit functions to the objects by passing the select set as an argument to the functions.

AXL-SKILL has functions to do the following:

- Set the Find Filter to control the types of objects selected and select options
- Select objects at a single point, over an area, or by name
- Select parts of objects (for example, pins, which are parts of symbols)
- Add or remove objects from the select set (the set of selected objects)
- Get `dbids` and return the count of `dbids` in the select set
- Add `dbids` to and remove them from the select set before using it.

See [Chapter 5, “Selection and Find Functions,”](#) for information about select set functions.

Note: PCB Editor highlights selected objects whenever it refreshes the display.

Design Files

Design files are containers for PCB Editor database objects. AXL-SKILL has two major design file types:

Layout	Contains printed circuit or MCM layout data.
Symbol	Contains the definition drawing of a symbol. The compiled output of a symbol file can be added to layouts. Symbol files can define any of package, mechanical, format, and shape symbols.

Logical Objects

Logical objects are the objects in the netlist that PCB Editor loads from a schematic or third-party netlist file:

Allegro PCB and Package User Guide: SKILL Reference

Introduction to PCB Editor SKILL Functions

Component	Contains the electrical functions, such as nand gates, and pins that define the electrical behavior of an object. A component's reference designator is its name.
Net	Is the set of all the etch objects—ppins, etch paths, shapes, and vias—associated with a particular signal name. Every net has a name which is its signal name. A net contains one or more branches. Each branch is a list of the etch objects that are physically connected among themselves. A branch can include ppins, etch paths, shapes, and vias. The number of branches in a net varies as PCB Editor connects or disconnects parts of the net. A completely connected net consists of one and only one branch.

Layer Attributes

Each PCB Editor figure exists on a class/subclass in the database. For example, a c-line might be on class ETCH, subclass TOP. AXL-SKILL represents this class/subclass combination with a layer attribute. Each AXL-SKILL layer is in one-to-one correspondence with an PCB Editor class/subclass combination. A later section describes this structure in detail.

PCB Editor Properties

Although they are a class of PCB Editor objects, you do not create or access PCB Editor properties directly. Rather, you use AXL-SKILL commands to attach, delete, and read the values of properties on PCB Editor database objects. You can also use AXL-SKILL commands to read, create and delete definitions of your own properties.

Property Definitions

AXL-SKILL stores each property definition for an object indirectly associated with that object. You can access any property definition on an object to find its value and you can create new, user-defined properties using AXL-SKILL functions.

You can use an AXL-SKILL function to attach a property to an object if the object accepts that property type. The property must be defined in the property dictionary of that database. A property definition is an object that contains the property name, value type, and a list specifying to what object types it can be attached.

Figures

The following AXL-SKILL figures are PCB Editor geometry types.

Arc	Is a figure that is either an arc or a circle. You can specify arcs to AXL-SKILL with start and end points, and either radius or center point. (PCB Editor figure: <code>arc segment</code>).
Branch	Is a collection of etch figures that make up one physically connected part of a net. Nets are made up of branches, and branches are made up of pins, vias, tees and etch figures, as described later in this chapter.
DRC	Is a design rule violation marker with one or more object identities and a violation type and location.
Line	An object defined by the coordinates of its center line and a width (PCB Editor figure: <code>line segment</code>).
Path	Is a sequence of end-to-end lines and arcs on the same layer. Each segment can have a different width (PCB Editor figures: <code>lines</code> and <code>clines</code>).
PPin	Is a physical instance of a pin with associated padstack.
Polygon	Is an unfilled, closed path (PCB Editor figure: <code>unfilled shape</code>).
Shape	Is a filled shape. It can optionally contain voids.
Pad	Is a geometric shape (circle, oblong, rectangle) defining the shape of one type of pad on one layer. Pads are always owned by padstacks.
Symbol	Is a collection of geometries and text with a type name, location, rotation and mirroring.
Tee	Is the single point where the endpoints of three or more etch paths connect.
Text	Is a string of characters with associated size, mirror, rotation, and location.

Allegro PCB and Package User Guide: SKILL Reference

Introduction to PCB Editor SKILL Functions

Via Is a connecting drill path between layers with associated padstack.

The following types are not figures but contain geometry that defines figure instances:

Padstack (Pin/Via Definition) Contains the definition data for all ppins or vias of a named type.

Symdef Contains the definition data for all symbols of a named type.

Accessing PCB Editor Colors with AXL-SKILL

You can access predefined colors and PCB Editor database colors using AXL-SKILL. Only graphics editors support access to PCB Editor database colors.

Forms

You set and access pre-defined colors by their symbols. The pre-defined colors include the following:

- 'black
- 'white
- 'red
- 'green
- 'yellow
- 'blue
- 'button

Button means grey, the color of buttons in the application.

Note: You can use only pre-defined colors in PCB Editor forms.

Design Object

Graphics editors support access to the colors used for PCB Editor layers. These are represented by integers.

Allegro PCB and Package User Guide: SKILL Reference

Introduction to PCB Editor SKILL Functions

AXL API calls, including `axlGetLayer` (“class/subclass”) or its primitive form `axlGetParm(paramLayerGroup:<class>/paramLayer:<subclass>)`, return the current color setting of a layer via the `color` attribute call, as shown.

```
p = axlGetLayer("etch/top")
p->color           ->2
```

These color settings range between 1 and 24 with 0 reserved for the background color.

Form based interfaces supporting color include the following:

- `axlFormDoc`
- `axlFormColorize`
- `axlFormGridDoc`
- `axlGRPDoc`

Notes:

- AXL does not allow you to change the red/green/blue (RGB) of PCB Editor database colors.
- Pre-defined colors are restricted to minimize problems with 8 bit color graphics on UNIX.

Database Objects

A design is made of various database objects that you can combine to make other database objects. This section describes the relationships among database objects.

Parts of a Design

A design can include any of the following database objects:

- Property Dictionary
- Lines
- Text
- Polygons
- Shapes
- Property Definitions
- DRCs

- Vias that are Padstack object types
- Symbols that are Symdef object types
- Components
- Nets

Parts of a Symbol

Symbols that are Symdef objects types can include any of the following database objects:

- PPins that are Padstack object types
- Vias that are Padstack object types
- Lines
- Arcs
- Text
- Polygons
- Shapes

Parts of a Branch

Branches can include any of the following database objects:

- Tees
- Vias that are Padstack object types
- PPins that are Padstack object types
- Paths
- Shapes

Parts of a Path

A path can include any of the following database objects:

- Lines
- Arcs

Parts of a Symdef

A symdef can include any of the following database objects:

- PPins that are Padstack object types
- Vias that are Padstack object types
- Lines
- Arcs
- Text
- Polygons
- Shapes

Types of Parameters

PCB Editor parameters store feature or board level options to the PCB Editor database. You can modify parameters using a SKILL program. The parameter types include the following:

- Design
- Display
- Layer Group
- Textblock Group

Table 1-1 Other Database Object Relationships

Object Name	Object Parts
Property Dictionary	Property Entries
Net	Branches
Padstack	Pads
Polygon	Paths
Shape	Paths
Void	Polygons
Polygon	Paths
PPin	Pin Number Text
Layer Group	Layers
Textblock Group	Textblocks

The PCB Editor Database User Model

Overview

This chapter describes each AXL database object type, listing each type's attributes and relationships to other object types.

Object Types

- Figure objects
 - Arcs ([Table 2-3 on page 68](#))
 - Branches ([Table 2-4 on page 68](#))
 - Design Files ([Table 2-5 on page 69](#))
 - Drcs ([Table 2-6 on page 69](#))
 - Lines ([Table 2-9 on page 71](#))
 - Paths ([Table 2-12 on page 73](#))
 - Polygons ([Table 2-14 on page 75](#))
 - Ppins ([Table 2-13 on page 73](#))
 - Shapes ([Table 2-16 on page 76](#))
 - Symbols ([Table 2-17 on page 77](#))
 - Tees ([Table 2-19 on page 78](#))
 - Vias ([Table 2-21 on page 79](#))
 - Pads ([Table 2-10 on page 72](#))
 - Padstacks ([Table 2-11 on page 72](#))
 - Symdefs ([Table 2-18 on page 78](#))
- Logical objects

- ❑ Components ([Table 2-24](#) on page 81)
- ❑ Functions ([Table 2-26](#) on page 82)
- ❑ Function Pins ([Table 2-27](#) on page 83)
- ❑ Nets ([Table 2-29](#) on page 85)
- Property dictionary objects ([Table 2-33](#) on page 89)
- Parameter objects
 - ❑ Design ([Table 2-36](#) on page 92)
 - ❑ Display ([Table 2-37](#) on page 93)
 - ❑ Layer Group ([Table 2-39](#) on page 94)
 - ❑ Layer ([Table 2-40](#) on page 95)
 - ❑ Textblock Group ([Table 2-41](#) on page 95)
 - ❑ Textblock ([Table 2-42](#) on page 96)

Description of Database Objects

Although a database object type can have dozens of attributes, you need only learn the semantics of a few attributes to get useful information from the PCB Editor database. Requesting an attribute not applicable to an object, or a property not existing on the object, causes the access function to return `nil`.

AXL Database Rules

The PCB Editor database interacts with AXL functions as specified in the following rules. Changes to the PCB Editor database made using AXL functions can affect both the database references (*dbids*) and the attributes of the PCB Editor database objects that the AXL program has already accessed.

- Invoking the `axlShell` function or editing a new PCB Editor database invalidates all *dbids*.

Accessing the object's attributes with an out-of-scope *dbid* yields unreliable values. The `axlDBRefreshId` function returns `nil` for any out-of-scope *dbid*.
- An AXL function that modifies one attribute of an object may cause a related attribute of that object to become out-of-date.

A parent's attribute may become out-of-date when you modify one of its children's attributes. For example, changing path width might affect the bounding box attribute of both a child and its parent.

- Operations you perform on an object affect the attributes of other objects if they refer to the changed object either directly or indirectly.

An example of direct reference is deleting a segment from a path. An example of indirect reference is changing the width of a single segment in a path. These can cause `isSameWidth` to be incorrect.

- Accessing an out-of-date `dbid` does not cause the AXL program to crash or corrupt the PCB Editor database.

- The AXL function, `axlDBRefreshID`, updates an object.

AXL does not update objects asynchronously.

- PCB Editor maintains properties across operations for all objects that support properties.

- DRC objects are volatile.

By changing PCB Editor database objects, you can create or destroy DRCs.

You cannot *directly* change a DRC object.

The following PCB Editor rule for treating non-etch figures applies only to paths or path segments:

- Path `dbids` on non-etch layers are more stable than those on etch layers.

Deleting a segment from a path breaks the segment into two paths with separate `dbids`. Non-etch paths never merge, even if they touch.

The PCB Editor database treats etch figures differently from non-etch figures. The following are the etch figure rules. [Figure 2-1](#) on page 64 shows the connectivity model used by PCB Editor.

- Path, line and arc `dbids` are volatile on etch layers.

PCB Editor merges and breaks these objects to maintain connectivity.

- Deleting a segment from a path so it detaches from a pin, via, tee, or shape causes the etch to be classed as *floating*.

That means the etch is not a member of any net. A floating etch has a `nil` netname.

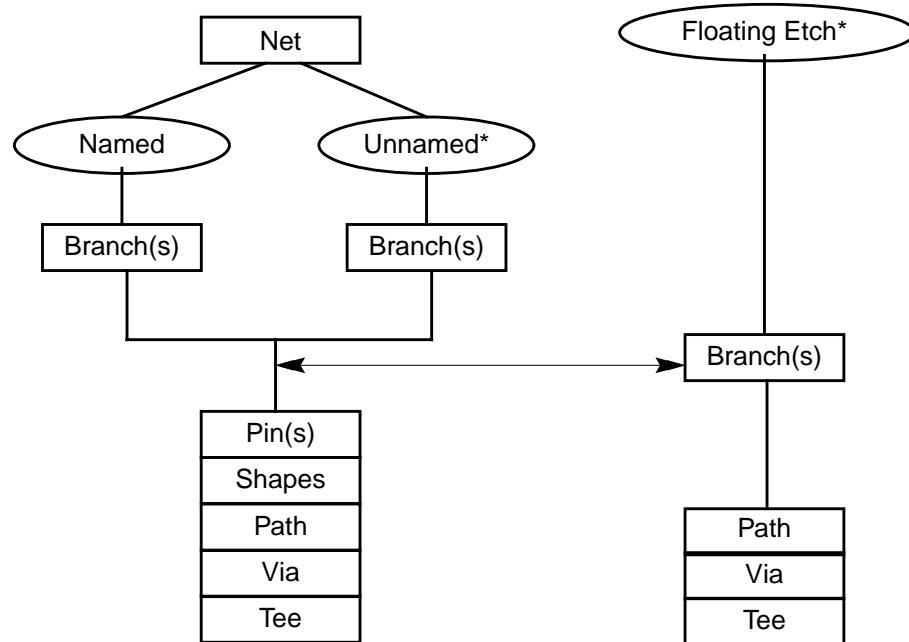
- Tees and branches are volatile.

Allegro PCB and Package User Guide: SKILL Reference

The PCB Editor Database User Model

Changes in paths or path segments can cause tees to disappear or branches to combine or break into multiple branches.

Figure 2-1 PCB Editor Connectivity Model



Tee	Connection of three or more paths, added and removed as needed
Via	Can be deleted by ripup or glossing operations
Path	The net attribute on paths, vias and tees changes due to connectivity to pins and shapes.
Line/Arc	Paths, lines and arcs can be broken, merged, or can change parent with the modification (add, delete or modify) of any etch object.

* The “net” attribute is an empty string “” for all figures that are on a dummy net.

Data Types

AXL database objects can have the following data types:

Type	Meaning
bbox	Boundary box (list of two points, lower left and upper right of a rectangular area that encloses the object)
integer	Signed integer number
float	Floating-point number
string	A string. For attributes with a list of possible string values, the attribute description lists the allowed values.
t/nil	Either true (t) or false (nil)
dbid	PCB Editor object identifier
l_dbid	List of <i>dbids</i>
point	A point—a list of two floats denoting a coordinate pair
l_propid	A list of properties accessed by axlDbGetProperties
l_fill	t = solid; nil = polygon; r_fill = crosshatch type

Generic PCB Editor Object Attributes

The following attributes are generic to all PCB Editor database objects. All PCB Editor database objects have at least these attributes.

Table 2-1 Generic Object Attributes

Attribute Name	Type	Description
objType	string	Name of the object type
prop	propid	Attached properties
parentGroups	l_dbid	List of groups to which the object belongs
readOnly	t/nil	t = cannot modify object with AXL function

propid refers to properties attached to the object. When a *dbid* is returned to an AXL program, the properties attached to that object are not immediately returned. You access the property value by referencing the property name via the prop attribute. The `axlDBGetProperties` function returns the property names/value pairs as an “assoc” list to allow easier processing by applications.

Figure Database Types

Figures, in PCB Editor, also share a common set of attributes. However, “figure” is not actually an attribute of any object. [Table 2-2](#) on page 67 lists the common figure attributes. In PCB Editor, figures are sometimes called geometries.

Among other attributes, figures have a bounding box called *bBox*. *bBox* is an orthogonal rectangle that defines the geometrical extents of the figure.

Table 2-2 Common Figure Attributes

Attribute Name	Type	Description
bBox	bbox	Figure's bounding box
branch	dbid	For etch, the figure's branch parent
layer	t_layer	Layer of figure, nil if object is multi-layer
parent	dbid	Nonconnective owner
net	dbid	Net object if figure is associated with a net

Note: The “*net*” attribute is an empty string “ ” for all figures that are on a dummy net.

Attributes for Each Figure Type

The following tables list the attributes specific to each PCB Editor object type. The Common Figure attributes (see [Table 2-2](#) on page 67) and the Generic PCB Editor Object attributes (see [Table 2-1](#) on page 66) also apply to these figure types.

The tables are in alphabetical order by figure type name. The attributes in each table are in alphabetical order by attribute name.

Table 2-3 Arc Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
isCircle	t/nil	t = circle; nil = unclosed arc
isClockwise	t/nil	t = clockwise; nil = counterclockwise
isEtch	t/nil	t = a CLINE; nil = a LINE
objType	string	Type of object, in this case "arc".
parent	dbid	Path, polygon or shape
radius	float	Radius
startEnd	l_point	Start and end points of arc
width	float	Width of arc
xy	point	Location of arc center

Table 2-4 Branch Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
children	l_dbid	List of <i>dbids</i> of the objects that make up branch: paths, tees, vias, pins and shapes
objType	string	Type of object, in this case "branch"
parent	dbid	Always nil

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-5 Design Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bus	l_dbid	List of busses
compdefs	l_dbid	List of component definitions
components	l_dbid	List of components
diffpair	l_dbid	List of diffpairs
drcs	l_dbid	List of DRCs
drcState	symbol	State of DRC t = up to date nil = out-of-date batch = batch out-of-date
ECsets	l_dbid	List of Electrical Csets
groups	l_dbid	List of groups
matchgroup	l_dbid	List of match groups in the design
module	l_dbid	List of module instances in the design
nets	l_dbid/nil	List of nets
objType	string	Type of object, in this case "design"
padstacks	l_dbid	List of padstacks
symbols	l_dbid	List of symbol instances
symdefs	l_dbid	List of symbol defs
xnet	l_dbid	List of Xnets (no nets with VOLTAGE property)

Table 2-6 DRC Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
actual	string	Actual value (user units)
expected	string	Expected value (user units)

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-6 DRC Attributes, continued

Attribute Name	Type	Description
fixed	t/nil	t = PCB Editor generated DRC nil = user defined DRC
objType	string	Type of object, in this case "drc"
parent	dbid	Design <i>dbid</i>
source	string	DRC source (property or constraint set name)
type	string	DRC type
violations	l_dbid	List of figures causing error (2 max)
xy	point	Location of DRC marker

Table 2-7 Group Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of members of the group
name	string	Name of the group
objType	string	Type of object, in this case "group"
type	string	Predefined group type.
Note: This cannot be defined in SKILL. User defined groups are considered "GENERIC."		

Table 2-8 Module Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of members of the module
name	string	Name of the module
objType	string	Type of object, in this case "group"
type	string	"MODULE"

Allegro PCB and Package User Guide: SKILL Reference
The PCB Editor Database User Model

Table 2-9 Line Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
isEtch	t/nil	t = a CLINE; nil = a LINE
lineType	s_type	symbol: horizontal, vertical, odd
objType	string	Type of object, in this case "line"
parent	dbid	Path, polygon, or shape
startEnd	l_point	Start and end points
width	float	Width of line

Table 2-10 Pad Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
figure	lr_path	List of <i>r_paths</i> defining pad's boundary <i>lr_path</i> always contains at most one <i>r_path</i> <i>nil</i> denotes a null pad
figureName	string	Name of pad figure is one of the following: CIRCLE, SQUARE, OBLONG, RECTANGLE, SHAPE or <i>nil</i> (if drill only)
flash	string	Flash name
layer	string	Pad layer
objType	string	Type of object, in this case "pad"
offset	l_point	Offset of pad (relative to pin/via origin)
parent	dbid	Padstack
type	string	Pad type is one of: REGULAR, ANTI, or THERMAL

Table 2-11 Padstack (PPin/Via Definition) Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
drillDiameter	float	Drill hole diameter
isThrough	t/nil	t = through padstack
name	string	Padstack name
objType	string	Type of object, in this case "padstack"
pads	ll_dbid	List of pads
parent	dbid	Design
startEnd	lt_layer	Start and end layer of padstack

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-12 Path Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid/nil	Branch owner
hasArcs	t/nil	t = path has one or more arcs
isSameWidth	t/nil	t = all segments in path have same width
isEtch	t/nil	t = a CLINE; nil = a LINE
nSegs	integer	Number of segments in path
objType	string	Type of object, in this case "path"
parent	dbid	Branch, symbol, shape or nil
segments	l_dbid	List of arc and line figures in this path
symbolEtch	dbid	Symbol owner if etch.

Table 2-13 Pin Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid/nil	Branch owner
component	dbid/nil	Component owner of pin nil if unassigned symbol pin
definition	dbid/nil	Padstack definition nil if unplaced component pin
fixedByTestPoint	t/nil	OBSOLETE - kept for backwards compatibility. Use axlDBIsFixed(<dbid>) or axlDBControl(?testPointFixed) instead.
functionPins	l_dbid/nil	List of function pins nil if unassigned symbol pin
isExploded	t/nil	t = pin is instance edited
isMech	t/nil	t = pin is mechanical

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-13 Pin Attributes, continued

Attribute Name	Type	Description
isMirrored	t/nil	<code>t</code> = pin is mirrored
isThrough	t/nil	<code>t</code> = pin is a throughhole
mirrorType	string	Type of mirror.
name	string	Padstack name of this pin <code>nil</code> if unplaced component pin
number	string	Pin number
objType	string	Type of object, in this case "pin"
pads	l_dbid	Unordered list of pads. ¹ To access a particular pad, use <code>ax1DBGetPad</code> .
parent	dbid	<i>dbid</i> of symbol owning this pin ² <code>nil</code> if pin is standalone (as it is in a symbol drawing)
relRotation	float	Pin rotation (relative to symbol)
relxy	point	Location (relative to symbol)
rotation	float	Pin rotation (absolute)
startEnd	lt_layer	Range of layers spanned by pin ²
testPoint	t_layer/nil	<i>t_layer</i> , denotes layer of testpoint <code>nil</code> = pin is not a testpoint
use	string	Pin use as shown by show element
xy	point	Location of pin in absolute coordinates

1. May be `nil` if component is unplaced.
2. Will say etch/(unknown) if unplaced component.

Note: Ppins straddle the line between physical and logical elements. They have attributes that are conditional on their owners. If the padstack definition is `nil`, then the pin is purely logical. If the component attribute is `nil`, then the pin is purely physical. If both are non-`nil`, then the pin is fully instantiated.

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-14 Polygon Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
area	float	Area of the polygon in drawing units.
bBox	bBox	Bounding box.
holes	list	List of <i>o_polygons</i> .
isHole	t/nil	t = polygon is a hole
isRect	t/nil	t = polygon is a rectangle
nSegs	integer	Number of segments in polygon
objType	string	Type of object, in this case "polygon"
parent	dbid	Symbol, shape (for voids), or nil
segments	l_dbid	Path describing boundary of shape. Boundary consists of line and arc segments.
symbolEtch	dbid	Symbol owner if etch
vertices	list	Outer boundary available as a list containing a point, which is the vertex of a polygon, and a floating point number, which is the radius of the edge from the previous to the present vertex.

Table 2-15 Rat_T Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
name	string	Name of T to T-<n>
net	dbid	Net of Rat-T
objType	string	Type of object, in this case "rat_t"
parent	dbid	Net
xy	point	Location of T

Table 2-16 Shape Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bBox	bBox	Bounding box
branch	dbid/nil	Branch owner
children	l_dbid	Used when a Boundary Shape points to a list of dynamic shapes
connect	l_dbid/nil	List of connected figures
fill	g_fill/t/nil	Fill pattern (see axlDBCreateOpenShape on page 125). <code>t = filled; nil = unfilled</code> Each <code>axlFillType</code> has spacing width, origin, and angle.
fillood	t/nil	Dynamic fill is out of date. <code>t = shape needs fill updating (Only dynamic shapes can be t)</code> <code>nil = shape does not refill</code>
holes	list	List of o_polygons
isHole	t/nil	<code>t = polygon is a hole</code> <code>nil = polygon is not a hole</code>
isRect	t/nil	<code>t = shape is a rectangle</code>
nSegs	integer	Number of segments in polygon
objType	string	Type of object, in this case "shape"
parent	dbid	Always nil
priority	integer/nil	If shape is a dynamic shape boundary this is an integer voiding priority. For all other shapes this is nil. The priority is relative to other dynamic shapes on the same layer. If two dynamic shapes are coincident, the shape with the higher priority wins in voiding. This number is re-calculated as needed, so use only for comparison purposes.
segments	l_dbid	Path boundary of shape

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-16 Shape Attributes, *continued*

Attribute Name	Type	Description
shapeAuto	l_dbid/nil	If dynamic shape list of generated shapes on the matching auto-gen ETCH layer
shapeBoundary	dbid/nil	If this shape is generated from a dynamic shape, this points to that shape.
shapelsBoundary	t/nil	This shape is a dynamic shape, for example, on BOUNDARY class.
symbolEtch	dbid	Symbol owner, if etch
vertices	list	Outer boundary available as a list containing a point (the vertex of a polygon) and a floating point number (the radius of the edge from the previous to the present vertex).
voids	l_dbid	List of polygon boundaries defining voids in this shape

Note: You cannot manipulate (move, add property, delete and more) auto-generated shapes (shapeBoundary != nil). You should modify the dynamic shape (shapeBoundary). You can use axlSetFindFilter to set the find filter to auto-select the boundary shape when the user selects one of the auto-generated children.

Table 2-17 Symbol Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
children	l_dbid/nil	List of figures other than pins making up symbol
component	dbid	Component owner of symbol
definition	dbid	Symbol definition
isMirrored	t/nil	t = symbol is mirrored
mirrorType	string	Type of mirror.
name	string	Symbol name
objType	string	Type of object, in this case "symbol"
parent	dbid	Design (no other parent possible for symbols)

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-17 Symbol Attributes, *continued*

Attribute Name	Type	Description
pins	l_dbid/nil	List of pins
refdes	string/nil	Reference designator
rotation	float	Symbol rotation
type	string	Symbol type is one of: PACKAGE, MECHANICAL, FORMAT or SHAPE
xy	point	Symbol location

Table 2-18 Symdef (Symbol Definition) Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
children	l_dbid/nil	List of figures other than pins making up shape
instances	l_dbid	Symbol instances
name	string	Name of symbol definition
objType	string	Type of object, in this case "symdef"
parent	dbid	Design
pins	l_dbid/nil	List of pins
type	string	Symbol type is one of the following: PACKAGE, MECHANICAL, FORMAT, SHAPE, or DRAFTING

Table 2-19 Tee Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid	Branch owner
objType	string	Type of object, in this case "tee"
parent	dbid	Branch
readOnly	t	User cannot directly modify

Allegro PCB and Package User Guide: SKILL Reference
The PCB Editor Database User Model

Table 2-19 Tee Attributes, *continued*

Attribute Name	Type	Description
xy	point	Location

Table 2-20 Text Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
isMirrored	t/nil	t = text mirrored
justify	string	"left", "right" or "center"
mirrorType	string	Type of mirror.
objType	string	Type of object, in this case "text"
parent	dbid	Symbol or nil
rotation	float	Rotation angle
text	string	The text itself
textBlock	string	Text block type
xy	point	Location of text origin

Table 2-21 Via Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid/nil	Branch owner
definition	dbid	Padstack definition
isMirrored	t/nil	t = via mirrored
isThrough	t/nil	t = through via
mirrorType	string	Type of mirror.
name	string	Padstack name
objType	string	Type of object, in this case "via"

Allegro PCB and Package User Guide: SKILL Reference
The PCB Editor Database User Model

Table 2-21 Via Attributes, *continued*

Attribute Name	Type	Description
pads	l_dbid	Unordered list of pads. To access a specific pad, use <code>ax1DBGetPad</code> .
parent	dbid	Symdef or nil
rotation	float	Via rotation
startEnd	lt_layer	Start and end layer of via
testPoint	t_layer/nil	Via test point state is one of: ETCH/TOP or ETCH/BOTTOM
xy	point	Location of via

Logical Database Types

PCB Editor logical database objects have these generic attributes (see [Description of Database Objects](#) on page 62): objType, prop, and readOnly. Logical database objects do not have other attributes in common. The tables below list the attributes for each PCB Editor logical type.

Table 2-22 Bus Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of xnets of the bus
name	string	Name of the bus
objType	string	“group”
type	string	“BUS”

Table 2-23 Compdef Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
class	string	Component classification
components	l_dbid	List of component instances of this definition.
deviceType	string	Device type of the component
functions	l_dbid	List of functions.
objType	string	Type of object, in this case “compdef”
pins	l_dbid	List of pins comprising the component.

Table 2-24 Component Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
class	string	Component classification

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-24 Component Attributes, continued

Attribute Name	Type	Description
compdef	dbid	dbid of component definition (COMPDEF)
deviceType	string	Device type of component
functions	l_dbid	List of functions
name	string	Reference designator
objType	string	Type of object, in this case "component"
package	string	Package name
pins	l_dbid	List of pins comprising component
symbol	dbid/nil	dbid of the placed symbol of this component, nil if unplaced

Table 2-25 Diffpair Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of xnets of the diffpair
name	string	Name of the diffpair
objType	string	"group"
type	string	"DIFFPAIR"
userDefined	t/nil	If you create t, can be modified. nil indicates creation by SigNoise models and cannot be changed.

Table 2-26 Function Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
name	string	Function designator
objType	string	Type of object, in this case "function"

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-26 Function Attributes

Attribute Name	Type	Description
parent	dbid	<i>dbid</i> of component owning this function
pins	l_dbid	List of function pins composing function
slot	string	Slot name
type	string	Function type

Table 2-27 Function Pin Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
name	string	Function pin name
objType	string	Type of object, in this case "functionPin"
parent	dbid	<i>dbid</i> of function owning this pin
pin	dbid	Pin owner of function pin
swap code	integer	Swap code of function pin
use	string	Pin usage description, one of UNSPECIFIED, POWER, GROUND, NC, LOADIN, LOADOUT, BI, TRU, OCA, OCL

Table 2-28 MATCH_GROUP Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of xnets, nets and pinpairs making up this group.
name	string	Name of the match group.
objType	string	"group"
pinpair	l_dbid	List of pinpairs associated with xnet
type	string	"MATCH_GROUP"

Allegro PCB and Package User Guide: SKILL Reference

The PCB Editor Database User Model

Note: For more information, see `axlMatchGroupCreate`.

Table 2-29 Net Attributes

Note:

Attribute Name	Type	Description
Also includes generic object attributes		
branches	l_dbid	List of branches
name	string	Net name
bus	dbid	Bus dbid if part of a bus
nBranches	integer	Number of branches (when exactly one, net is fully connected)
objType	string	Type of object, in this case "net"
pinpair	l_dbid	List of pinpairs associated with net (1)
Note: If a net is a member of an xnet, all pinpairs appear on the xnet.		
ratsnest	l_dbid	List of ratsnest for net
ratT	l_dbid	List of rat_T's. If none exist, this is NULL.
rpd	l_rpd	List of lists for each member net of a match group (mg_dbid t_relatePropDelay) For more information, see axlMatchGroupCreate
scheduleLocked	t/nil	t = net schedule cannot be changed

Table 2-30 Pinpair Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
ECset derived	t/nil	If t , pinpair was created from an ECset. Nil indicates pinpair created due to net override property.
groupMembers	l_dbid	List of two pins making up pinpair.
name	string	Name of the pinpair (<refdes>. <pin#>: <refdes>. <pin#>)
objType	string	"group"

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-30 Pinpair Attributes, continued

Attribute Name	Type	Description
parent	l_dbid	Net or xnet owning the pinpair.
parentGroups	l_dbid	Lists match groups that have this pinpair. May also list other parent groups.
rpd	l_rpd	For each match group that has this pinpair as a member, will list as a list of lists. (mg_dbid t_relatePropDelay)
type	string	"PIN_PAIR"

Note: For nets that are part of an xnet, the pinpair always has the xnet as the pinpair owner. For more information on the rpd attribute, see `axlMatchGroupCreate`.

Table 2-31 RATSNEST Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
bus	t/nil	Currently being shown with bus routes option
objType	string	Type of object, in this case "ratsnest"
pinsConnected	t/nil	Ratsnest not displayed (both pins on same branch)
pins	l_dbid	The two pins (or ratTs) that the rats connect
pwrAndGnd	t/nil	Net is power and ground scheduled
ratnest	t/nil	Two dbids of the next ratsnest for dbid's of the pins attribute.
ratsPlaced	t/nil	User defined rats only, one or more pins unplaced
userDefined	t/nil	Ratsnest is user defined

Table 2-32 Xnet Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
bus	dbid	Bus dbid if part of a bus.
diffpair	dbid	Diffpair dbid if part of a diffpair.

Table 2-32 Xnet Attributes, continued

Attribute Name	Type	Description
groupMembers	l_dbid	List of nets of the xnet.
name	string	Name of the xnet.
objType	string	“group”
pinpair	l_dbid	List of pinpairs associated with xnet.
rpd	l_rpd	For each match group tha has this xnet as a member, lists as a list of lists (<code>mg_dbid t_relatePropDelay</code>).
type	string	“XNET”.

Note: You can only define this attribute indirectly from the SigNoise model assignment. For more information on rpd, see `axlMatchGroupCreate`.

Property Dictionary Database Types

The following section contains tables listing the attributes of PCB Editor property dictionary objects. You must enter a dictionary object with a particular name in the property dictionary before you attach properties by that name to PCB Editor objects.

Object Types Allowing Attachment of Properties

You can attach properties to the database object types listed here. Each property type has a list of the objects types to which it can be attached. Attempting to attach a non-qualified property to an object returns `nil`.

NETS	COMPONENTS	FUNCTIONS	PINS
VIAS	SHAPES	SYMBOLS	CLINES
LINES	DRCS	FIGURES	DESIGNS
COMPDEFS	PINDEFS	FUNCDEFS	

Allowed Property Data Types

An PCB Editor property value can be one of the data types listed. The right column shows the checks you can optionally apply to user input for that data type. Pre-defined properties have pre-defined range checks. You define your own range checks for user-defined properties.

Note: The right column includes default units for that property data type, however, you can set the units of these standard data types in the `<tools>/text/units.dat` file.

Data Type	Data Check Available (default units)
STRING	N/A
INTEGER	range and units
REAL	range and units
DESIGN_UNITS	range (units of current design)
BOOLEAN	N/A
ALTITUDE	range (meters)
CAPACITANCE	range (pF)
DISTANCE	range (cm)
ELEC_CONDUCTIVITY	range (mho/cm)
LAYER_THICKNESS	range (mil)
IMPEDANCE	range (ohm)
INDUCTANCE	range (nH)
PROP_DELAY	range (nS)
RESISTANCE	range (ohm)
TEMPERATURE	range (degC)
THERM_CONDUCTANCE	range (w/cm-degC)
THERM_CONDUCTIVITY	range (w/cm-degC)
VOLTAGE	range (mV)
VELOCITY	range (m/sec)

Table 2-33 Property Dictionary Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
dataType	string	Data type for property value (see Allowed Property Data Types on page 88)
name	string	Name of property
objType	string	Type of object, in this case "propDict"
range	If_range	Optional limits for value
units	string	Optional value units
useCount	integer	Number of objects using property
write	t/nil	t = writable or user defined nil = read-only or PCB Editor pre-defined

Parameter Database Types

The following PCB Editor parameter types, which are critical to the function of interactive commands, are modeled:

- drawing
- layer
- text block
- display

Only drawing parameters support attached properties.

You can access and change parameters with `axlParamGetByName` and `axlParamSet`, respectively. See [Chapter 4, “Parameter Management Functions”](#) for functions that provide easier access to layers and other parameter objects.

Allegro PCB and Package User Guide: SKILL Reference

The PCB Editor Database User Model

Unlike other PCB Editor objects these attributes contain a column, Set?, that shows whether you can modify this field via the axlParamSet function.

Table 2-34 Artwork Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
groupMembers		l_string	List of film names
nChildren		number	Number of films
objType		string	Type of object, in this case "artwork"

Table 2-35 Artwork Film Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
drawMissingPadApertures	t/nil		Specifies the apertures you can use to draw (fillin) the shape of any pad for which there is no matching pad aperture in art_aper.txt. Not selecting this option means that you cannot draw such pads.
fullContact	t/nil		Applies to negative film. When t, a pin or via that is connected to a shape uses no flash, causing a solid mass of copper to cover the pad. When you do not select this field, a pin or via connected to a shape uses a thermal-relief flash.
groupMembers		l_string	List of layers comprising film
mirrored	t/nil		Specifies whether to mirror the photoplot output.
name		string	Name of film
negative	t/nil		Is film positive (nil) or negative (t)
objType		string	Type of object, in this case "artwork"

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-35 Artwork Film Parameter Attributes, *continued*

Attribute Name	Set?	Type	Description
offset		point	Specifies the x and y offset to add to each photoplot coordinate. If you enter positive x and y offsets, all photoplotted lines shift in the positive direction on the film.
rotation		integer	Rotation of the plotted film image. Choices are: 0, 90, 180, and 270 degrees.
shapeBoundingBox		float	Applies to negative film. Adds another outline around the design outline, extending the shape boundary of the filled area. This new artwork outline extends, by default, 100 mils in all directions beyond the design outline.
suppressShapeFill		t/nil	Area outside the shapes is not filled on a negative film. You must replace the filled areas with separation lines before running artwork.
suppressUnconnectPads		t/nil	Specifies that the pads of pins and vias with no connection to a connect line in a gerber data file are not plotted. This option applies only to internal layers and to pins whose padstack flags the pads as optional. Selecting this button also suppresses donut antipads in raster based negative artwork.
undefineLineWidth		float	Determines the width of any line that is 0.
useApertureRotation		t/nil	Specifies whether or not to use the aperture rotation raster. Artwork uses gerber behavior to determine which type of pad to flash.

Table 2-36 Design Parameter Attributes

Attribute Name	Set?	Type	Description
Includes generic object attributes			
accuracy	no	integer	Number of decimal places of accuracy
bBox	no	bbox	The design's bounding box
height	no	float	Height in user units
objType	no	string	Type of object, in this case "paramDesign"
units	no	string	Type of user units (mils, inch, micron, millimeter and centimeter)
width	no	float	Width in user units
xy	no	point	Lower left corner of design

Notes:

- To avoid harmful side effects, such as rounding errors, do not toggle between English and metric.
- Changes to accuracy are very time consuming since all objects in the design must be changed.

Setting drawing accuracy to a value greater than PCB Editor supports is an error. Typical range is 0 to 4.

- Change units and accuracy at the same time to avoid loss of accuracy.
- The size (width and height) may not be decreased where it will leave some objects outside of the drawing extents.

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-37 Display Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
activeLayer	yes	string	Active layer name
altLayer	yes	string	Alternative layer name which must be of group "etch"
objType	no	string	Type of object, in this case "paramDisplay"

Table 2-38 ECset Parameter Attributes

Attribute Name	Set?	Type	Description
Includes generic object attributes			
locked		t/nil	Cset locked from UI based editing
members		l_dbid	List of electrical constraints in set
name		string	Name of ECset
objType		string	Type of object, in this case "ecset"
prop		l_dbid	List of user defined properties on ECset
readOnly		t/nil	Cannot be modified, always t
topology		t/nil	This is derived from a topology file, and may contain constraints that have restrictions on how they can be modified.
prop		l_dbid	List of user-defined properties on an ECset.

Allegro PCB and Package User Guide: SKILL Reference
The PCB Editor Database User Model

Figure 2-2 PCB Editor Class/Subclass to AXL Layer Model

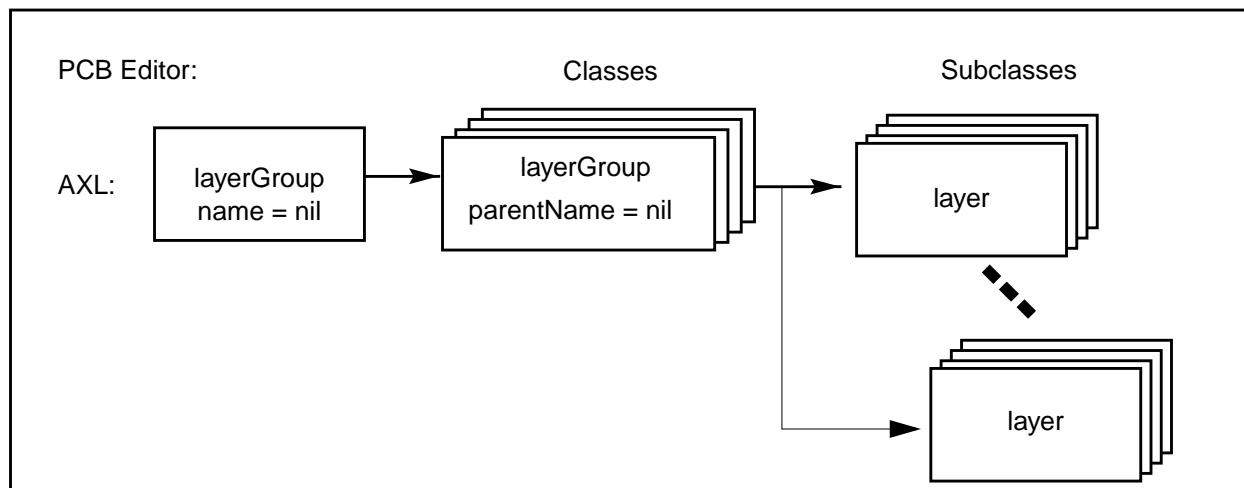


Table 2-39 Layer Group Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
color	yes	integer	Layer color index; -1 if all child layers are not the same color
groupMember	no	I_string	Names group members
isEtch	no	t/nil	Is an etch layer
name	no	string	Name of this layer (<code>nil</code> if class group holder)
nChildren	no	integer	Number of children
objType	no	string	Type of object, in this case " <code>paramLayerGroup</code> "
parentName	no	string	Name of parent (<code>nil</code> if at class level)
visible	yes	t/nil	All child layers are visible

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-40 Layer Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
color	yes	integer	Color index of layer
drcPhotoType	no	l_string	"Positive" or "negative" (applies only to etch)
isEtch	no	t/nil	Is an etch layer
name	no	string	Name of this layer
nextLayer	no	string	Next sub-layer in layer group (applies only to etch)
objType	no	string	Type of object, in this case "paramLayer"
parentname	no	string	Name of parent
visible	yes	t/nil	Layer is visible

Note: The PCB Editor class/subclass system is modeled in AXL as layers. The previous drawing figure shows how AXL layers are mapped to PCB Editor class/subclasses. PCB Editor's define etch form does not match this model. AXL does not support access to the dielectric pseudo-layers nor does it support analysis layer attributes such as material and thickness. To access these records, use PCB Editor's technology file feature.

Table 2-41 Textblock Group Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
groupMembers	no	l_string	Names of members in this group
nChildren	no	integer	Number of children
objType	no	string	Type of object, in this case "paramTextGroup"

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-42 Textblock Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
charSpace	no	float	Spacing between characters
height	no	float	Height of character
lineSpace	no	float	Spacing between lines
name	no	string	Name of block (currently 1 through 16)
objType	no	string	Type of object, in this case "paramTextBlock"
photoWidth	no	float	Width of character for photoplotting
width	no	float	Width of character

Table 2-43 Testprep Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
allowUnderComp	yes	t/nil	Allows test pads under components.
autoInsert	yes	t/nil	Allows automatic generation of test points as needed.
bareBoard	yes	t/nil	If nil, you can only test component pins on noncomponent design side. If t, then you can check pins on either side of the design, as long as padstack is defined on that side.
directTest	yes	t/nil	Allows pins to be selected as test point.
executelnc	yes	t/nil	If t, returns in incremental mode where you can analyze only nets without test points. If nil, removes all test points at beginning of run.
layer	yes	symbol	Layer for test: top, bottom or either.
maxTestDisplacement	yes	float	Maximum distance from pin or via where you can place the test point.

Allegro PCB and Package User Guide: SKILL Reference
 The PCB Editor Database User Model

Table 2-43 Testprep Parameter Attributes, continued

Attribute Name	Set?	Type	Description
minPadSize	yes	float	Minimum padstack size. Works with replaceVias to upscale padstacks.
minTestDisplacement	yes	float	Minimum distance from pin or via where you can place a test point. A value of 0 indicates DRC distance you should use.
minTestSpacing	yes	float	Minimum spacing between test points.
objectType	no	string	Type of object, in this case "testprep"
pinType	yes	symbol	Type of pin selectable for testing: input, output, pin, via, or point.
replaceVias	yes	t/nil	If t , replaces vias that are too small with a larger via.
testGridX	yes	float	Testprep grid.
testGridY	yes	float	Testprep grid.
testMethod	yes	symbol	Method used for test: single, node, or flood.
testPad	yes	nil/string	Padstack that you use for test pads on the probe side of the design. Must meet criteria specified in the <i>testPadType</i> attribute. If relative name is given, uses database then PSMPATH to find pad.
testPadDB	no	dbid	<i>dbid</i> of the testPad
testPadType	yes	symbol	Type of padstacks for probes: smd, through, or either.
testVia	yes	nil/string	Padstack that you use for testpads on the probe side of the design. Must be through hole. If relative name is given, uses database then PSMPATH to find pad.
testViaDB	no	dbid	<i>dbid</i> of testVia.
unusedPins	yes	t/nil	Allows test points on pins, not on a net.
The following are Text Controls			
alpha	yes	t/nil	If t, use Alphabetic extension, nil use numeric extension.

Table 2-43 Testprep Parameter Attributes, continued

Attribute Name	Set?	Type	Description
displayText	yes	t/nil	Displays text for each test point.
textOffsetX	yes	float	X offset of text from pad center.
textOffsetY	yes	float	Y offset of text from pad center.
textRotation	yes	integer	Rotation of text labels: values are 0, 90, 180, or 270 degrees. Other values are moduled and truncated.

Note: Specifying testVia or testPad loads them into the current database if they are not already loaded. Changing to another padstack does not delete the old padstack from the database.

Example

```
p = axlGetParam( "testprep" )
p->displayText = t
p->testMethod = 'flood
axlSetParam(p)
```

Turns on text display and sets test method to flood.

Database Create Functions

Overview

This chapter describes the AXL functions that add objects to the PCB Editor database. Some functions require input that you set up using available auxiliary functions, which are also described in this chapter. For example, PCB Editor paths consist of any number of contiguous line and arc segments. To add this multi-structure to the PCB Editor database, first create a temporary path, adding each line or arc segment using separate function calls. Once the temporary path contains all required segments, create the PCB Editor line-object, shape or void by calling the appropriate database create function, giving the path structure as an argument. The chapter shows several examples of the process.

Database create (`DBCCreate`) functions modify the active PCB Editor database in virtual memory and require a database save to make changes permanent in the file.

Supply all coordinates to these functions in user units, unless otherwise noted.

The functions described here do not display the objects immediately as they create them. To display all changes, call an interactive function, exit SKILL, or return control to the PCB Editor command interpreter.

- To immediately display an object you have just created, do one of the following:
 - Call the function `axlDisplayFlush`
 - or—
 - Call an interactive function

If you create an object and then delete it without calling `axlDisplayFlush` or calling an interactive function, the object never appears in the display.

The class of geometric objects that `DBCCreate` functions create are called *figures*. `DBCCreate` functions return, in a list, the *dbids* of any figures they create and a Boolean value `t` if the creation caused any DRCs. The functions return `nil` if they could not create

any figures. The exact structure of the data returned differs among the commands. See the individual commands for detailed descriptions.

You can set the active layer (PCB Editor class/subclass) by calling the `axlSetCurrentLayer` function. This function returns a `nil` if you try to set an invalid layer or if you try to create a figure on a layer that does not allow that figure type.

AXL-SKILL creates a figure as a member of a net only if the figure is on an etch layer. Where a function has a netname as an argument, and the active layer is an etch layer, the function attaches the figure to the net specified by that netname. If the net does not exist, an error occurs. If you specify `nil` for the netname, the function determines the net for the figure by what other figure it touches. If the figure is free standing, that is, touches nothing, the figure becomes a member of the dummy net (no net).

The functions use defaults for all parameters you do not supply. If you do not supply a required parameter (one without a default, for example, `pointList`) the function considers the call an error and returns `nil`.

The database create functions do not add figures to the select set. They leave the select set unchanged.

Path Functions

An PCB Editor `line` is a figure consisting of end-to-end straight line and arc segments, each segment having a width you can define separately. PCB Editor `shapes` and `polygons` are figures that define an area. A shape owns a closed line figure that defines the perimeter of the shape. The shape has an associated fill pattern and can also own internal `voids`. Each void in turn owns a polygon that defines its boundary.

A `path` is a set of contiguous arc and single straight line segments. In AXL, you first create a path consisting of the line and arc segments by adding each segment with a separate AXL function, then creating the actual figure using the appropriate `axlDBCreate` function, with the path as one of the arguments. With AXL convenience functions described later in this chapter, you can create rectangles, circles, and lines consisting only of straight segments.

All coordinate arguments to the path functions are in user units and are absolute to the layout origin.

Example

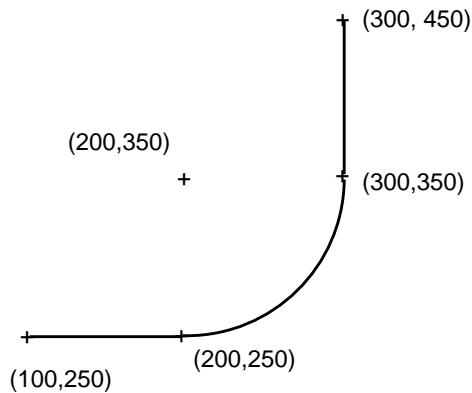
This general example shows how to create a path, then use it as an argument in an `axlDBCreate` function. The example creates a path consisting of a straight line segment,

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

then adds an arc and another line segment, and uses it as an argument to create a path that is a member of net "net1" on etch subclass "top":

```
path = axlPathStart( (list 100:250))
    axlPathLine( path, 0.0, 200:250 )
    axlPathArcCenter( path, 0.0, 300:350, nil, 200:350 )
    axlPathLine( path, 0.0, 300:450 )
axlDBCreatePath( path, "etch/top", "net1" )
```



axlPathStart

```
axlPathStart(  
    l_points  
    [f_width]  
)  
⇒ r_path/nil
```

Description

Creates a new path with a startpoint and one or more segments as specified by the list *l_points* and returns the path *dbid*. You can add more straight-line and arc segments to the returned *r_path* using the `axlPathArc` and `axlPathLine` functions described in this section. Once *r_path* has all the segments you require, create the actual database figure using the appropriate `axlDBCreate` function, with *r_path* as one of the arguments.

Arguments

<i>l_points</i>	List of <i>n</i> vertices, where <i>n</i> > 1. If <i>n</i> = 1, <i>r_path</i> returns with that single vertex as its startpoint, but with no segments. You must subsequently add at least one segment before adding it to the database If <i>n</i> > 1, <i>r_path</i> returns with <i>n</i> -1 straight-line segments.
<i>f_width</i>	Width for all segments, if any created, between the <i>l_points</i> . <i>f_width</i> is the default width for all additional segments added to <i>r_path</i> using <code>axlPath</code> functions. You can override this default width each time you add a segment using an <code>axlPath</code> function by using a <i>f_width</i> argument when you invoke the function.

Value Returned

r_path/nil Returns the *r_path* handle.

Note: This is a handle object, but is *not* an PCB Editor *dbid*.

Example

See start of the [Path Functions](#) on page 100.

axlPathArcRadius

axlPathArcAngle

axlPathArcCenter

```
axlPathArcRadius(
    r_path
    f_width
    l_end_point
    g_clockwise
    g_bigarc
    f_radius
)
⇒ r_path/nil

axlPathArcAngle(
    r_path
    f_width
    l_end_point
    g_clockwise
    f_angle
)
⇒ r_path/nil

axlPathArcCenter(
    r_path
    f_width
    l_end_point
    g_clockwise
    l_center
)
⇒ r_path/nil
```

Description

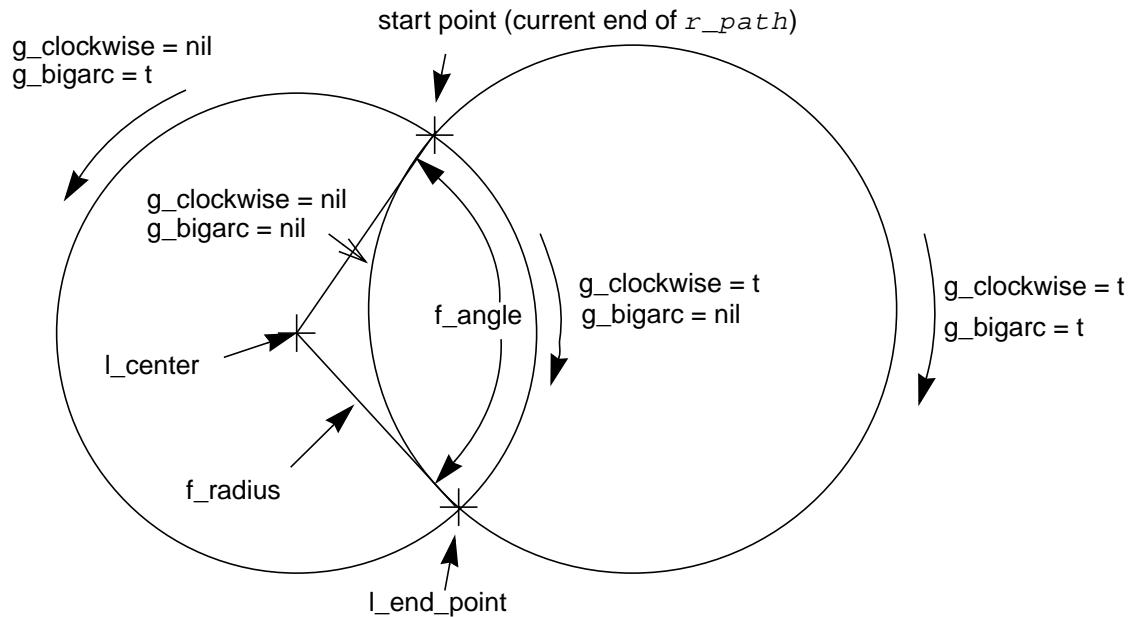
Each of these functions provides a way to construct an arc segment from the current endpoint of *r_path* to the given *l_end_point* in the direction specified by the Boolean *g_clockwise*, as described below and shown in [Figure 3-1](#) on page 106.

Attempts to create small arcs using many decimal points of accuracy may fail due to rounding errors.

Arguments

<i>r_path</i>	Handle of an existing <i>r_path</i> to receive arc segment.
<i>f_width</i>	Width of an arc segment in user units. Overrides, for this segment only, any width originally given in <code>axlPathStart</code> ; <code>nil</code> = use current width
<i>l_end_point</i>	End point to which an arc is to be constructed. Start point is the last point currently in <i>r_path</i> in absolute coordinates.
<i>g_clockwise</i>	Direction to create arc: <code>t</code> → create arc clockwise from start to endpoint <code>nil</code> → create counterclockwise. Default is counterclockwise (See Figure 3-1 on page 106).
<i>g_bigarc</i>	<code>axlPathArcRadius</code> : Create an arc greater than or equal 180 degrees (See Figure 3-1 on page 106).
<i>f_radius</i>	<code>axlPathArcRadius</code> : Arc radius in user units.
<i>f_angle</i>	<code>axlPathArcAngle</code> : Angle in degrees subtended by arc (See Figure 3-1 on page 106).
<i>l_center</i>	<code>axlPathArcCenter</code> : Arc center point in absolute coordinates.

Figure 3-1 Effects of *axlPathArc* Arguments



Value Returned

r_path Current path handle.

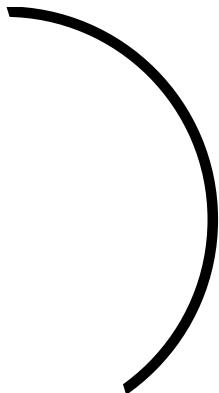
nil Arc path not created.

Example 1

```
mypath = axlPathStart( list( 8900:4400))
axlPathArcRadius( mypath, 12., 8700:5300, nil, nil, 500)
axlDBCreatePath( mypath, "etch/top")
```

Adds a smaller-than-180 degree counterclockwise arc by radius.

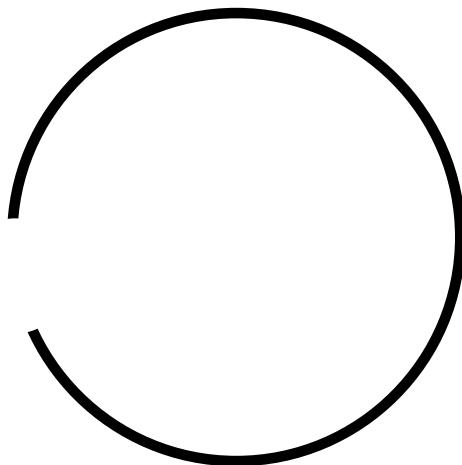
Creates the smaller possible arc:



Example 2

```
mypath = axlPathStart( list( 8900:4400 ))
axlPathArcAngle( mypath, 12., 8700:5300, nil, 330)
axlDBCreatePath( mypath, "etch/top" )
```

Adds a counterclockwise arc subtending 330 degrees.



Example of axlPathArcCenter

See [Example](#) on page 100.

axlPathLine

```
axlPathLine(  
    r_path  
    f_width  
    l_end_point  
)  
⇒ r_path/nil
```

Description

Adds a single straight line segment to the end of an existing *r_path* structure as specified by the arguments. Start point of the line is the last point in *r_path*.

Arguments

<i>r_path</i>	Handle of an existing path.
<i>f_width</i>	Width of the segment. <i>nil</i> = segment takes the width given when <i>r_path</i> was created.
<i>l_end_point</i>	End point of the line segment in absolute coordinates.

Value Returned

<i>r_path</i>	Path structure following addition of single straight line segment to end of <i>r_path</i> structure.
<i>nil</i>	No line segment added to <i>r_path</i> structure.

Example

See start of the [Path Functions](#) on page 100.

axlPathGetWidth

```
axlPathGetWidth(  
    r_path  
)  
⇒ f_width/nil
```

Description

Gets the default width of an existing path structure.

Arguments

r_path Handle of an existing path structure.

Value Returned

f_width Default width of the path structure.

nil *r_path* is not a path, or is empty.

Example

`axlPathGetWidth` returns the default path width of 173 mils.

```
path = axlPathStart( (list 1000:1250), 173)  
    axlPathLine( path, 29, 2000:1250)  
    axlPathLine( path, 33, 3000:3450)  
    axlDBCreatePath( path, "etch/top")  
axlPathGetWidth( path)  
⇒ 173.0
```

- Creates a path with width 173 mils
- Adds line segments at widths 29 and 33 mils

axlPathSegGetWidth

```
axlPathSegGetWidth(  
    r_pathSeg  
)  
⇒ f_width/nil
```

Description

Gets the width of a single segment in a path structure.

Arguments

r_pathSeg Handle of a segment of a path structure.

Value Returned

f_width Returns the width of the segment.

nil *r_pathSeg* is not a segment.

Example

`axlPathSegGetWidth` returns the width of that segment only, 33 mils.

```
path = axlPathStart( (list 1000:1250), 173)  
      axlPathLine( path, 29, 2000:1250)  
      axlPathLine( path, 33, 3000:3450)  
  
lastSeg = axlPathGetLastPathSeg(path)  
axlPathSegGetWidth( lastSeg)  
⇒ 33.0
```

- Creates a path with default width 173 mils
- Adds line segments at widths 29 and 33 mils
- Gets the last segment added with `axlPathGetLastPathSeg`

axlPathGetPathSegs

```
axlPathGetPathSegs(  
    r_path  
)  
⇒ r_pathList/nil
```

Description

Gets a list of the segments of a path structure, in the order they appear in the path.

Arguments

r_path Handle of an existing path structure.

Value Returned

r_pathList Returns a list of the segments of the path.

nil *r_path* is not a path.

Example

```
mypath = axlPathStart( (list 1000:1250), 173)  
    axlPathLine( mypath, 29, 2000:1250)  
    axlPathArcCenter( mypath, 12, 3000:2250, t, 3000:2250)  
mysegs = axlPathGetPathSegs( mypath)  
print mysegs  
⇒(array[6]:1057440 array[6]:1057416 array[6]:1057392)
```

- Creates a path
- Gets the segments of the path
- Prints the segments of the path

axlPathGetLastPathSeg

```
axlPathGetLastPathSeg(  
    r_path  
)  
⇒ r_pathList/nil
```

Description

Gets the last segment of a path structure.

Arguments

r_path Handle of an existing path structure.

Value Returned

r_pathList Returns the last segment of the path.

nil *r_path* is not a path.

Example

`axlPathSegGetWidth` returns the width of that segment only, 33 mils.

```
path = axlPathStart( (list 1000:1250), 173)  
      axlPathLine( path, 29, 2000:1250)  
      axlPathLine( path, 33, 3000:3450)  
  
lastSeg = axlPathGetLastPathSeg(path)  
axlPathSegGetWidth( lastSeg)  
⇒33.0
```

- Creates a path with the default width 173 mils
- Adds line segments at widths 29 and 33 mils
- Gets the last segment added using `axlPathGetLastPathSeg`

axlPathSegGetEndPoint

```
axlPathSegGetEndPoint(  
    r_pathSeg  
)  
⇒ l_endPoint/nil
```

Description

Gets the end point of an existing path structure.

Arguments

r_pathSeg Handle of a path segment.

Value Returned

l_endPoint List containing the end point of the path structure.

nil *r_pathSeg* is not the *dbid* of a path segment, or the structure is empty.

Example

```
path = axlPathStart( (list 1000:1250), 173)  
      axlPathLine( path, 29, 2000:1250)  
      axlPathLine( path, 33, 3000:3450)  
lastSeg = axlPathGetLastPathSeg(path)  
axlPathSegGetEndPoint( lastSeg)  
⇒(3000 3450)
```

- Creates a path with default width 173 mils
- Adds line segments at widths 29 and 33 mils
- Gets the last segment added with `axlPathGetLastPathSeg`

`axlPathSegGetWidth` returns the width of that segment only, 33 mils.

axlPathSegGetArcCenter

```
axlPathSegGetArcCenter(  
    r_pathSeg  
)  
⇒ l_point/nil
```

Description

Gets the center point of a path arc segment.

Arguments

r_pathSeg Handle of a path arc segment.

Value Returned

l_point List containing the center coordinate of the arc segment.

nil Segment is not an arc.

Example

`axlPathSegGetArcCenter` gets the center of the last arc segment.

```
path = axlPathStart( (list 1000:1250), 173)  
    axlPathLine( path, 29, 2000:1250)  
    axlPathArcCenter( path, 12., 3000:2250, nil, 2000:2250)  
  
lastSeg = axlPathGetLastPathSeg(path)  
axlPathSegGetArcCenter( lastSeg)  
⇒(2000 2250)
```

- Creates a path with a straight line segment and an arc segment
- Gets the last segment added with `axlPathGetLastPathSeg`

axlPathSegGetArcClockwise

```
axlPathSegGetArcClockwise(  
    r_pathSeg  
)  
⇒ t/nil
```

Description

Gets the clockwise flag (`t` or `nil`) of a path segment.

Arguments

`r_pathSeg` Handle of a path segment.

Value Returned

`t` Segment is clockwise.

`nil` Segment is counterclockwise.

Example

`axlPathSegGetArcCenter` returns `t`, meaning the arc segment is clockwise.

```
path = axlPathStart( (list 1000:1250), 173)  
    axlPathLine( path, 29, 2000:1250)  
    axlPathArcCenter( path, 12., 3000:2250, t, 2000:2250)  
  
lastSeg = axlPathGetLastPathSeg(path)  
axlPathSegGetArcClockwise( lastSeg)  
⇒ t
```

- Creates a path with a straight line segment and a clockwise arc segment
- Gets the last segment added using `axlPathGetLastPathSeg`

axlPathStartCircle

```
axlPathStartCircle(  
    l_location  
    f_width  
)  
⇒ r_path/nil
```

Description

Creates an axlPath structure (*r_path*) for a circle.

Arguments

l_location	Center and radius as ((X Y) R).
f_width	Edge width of the circle.

Value Returned

<i>r_path</i>	<i>r_path</i> with the circle as the only segment.
nil	axlPath structure not created.

Note: Width must be specified for this interface (it may be 0.0) and since it uses standard SKILL arg check, it must be a flonum.

Example

```
(axlPathStartCircle (list 100:200 20),0) ; no width specified.
```

axlDBCreatePath

```
axlDBCreatePath(  
    r_path  
    [t_layer]  
    [t_netName] / [‘line]  
    [o_parent]  
)  
⇒ l_result/nil
```

Description

Creates a path figure (line or cline) as specified.

Notes:

- axlDBCreatePath does not add a net name to an etch when the etch is not connected to a pin, via, or shape.
- If an etch is added, it is tied to the first net it touches, otherwise it remains “not on a net”.

Arguments

<i>r_path</i>	Existing path consisting of the straight-line and arc segments previously created by axlPath functions
<i>t_layer</i>	Layer on which to create a path figure. Default is the active layer.
<i>t_netName</i>	Name of the net to which the path figure is to belong. axlDBCreatePath ignores <i>t_netName</i> if <i>t_netName</i> is non-nil and <i>t_layer</i> is not an etch layer. If the net <i>t_netName</i> does not exist, axlDBCreatePath does not create any path, and returns nil.
‘ <i>line</i>	Changes default path created on an etch layer from a cline to a line.
<i>o_parent</i>	<i>dbid</i> of object to be the parent of the path figure. Use the symbol instance or use nil to specify the design. If you attach etch figures to a symbol parent, the figures are not associated with the symbol, and do not move with it.

Value Returned

<i>l_result</i>	List: (car) list of <i>dbids</i> of all path figures created or modified (cadr) t if DRCs are created. nil if DRCs are not created.
nil	Nothing was created.

Example

```
path = axlPathStart( list 100:0 100:500))

; create path on current default layer
axlDBCreatePath(path)

; create a cline path on top etch layer and assisgn to GND
axlDBCreatePath(path "ETCH/TOP" "gnd")

;create a line path on top etch layer
axlDBCreatePath(path "ETCH/TOP" 'line)

;have user create a two pick path on board geometry outline
axlDBCreatePath( axlEnterPath() "BOARD GEOMETRY/OUTLINE" )
```

axlDBCreateLine

```
axlDBCreateLine(  
    l_points  
    [f_width]  
    [t_layer]  
    [t_netname]  
    [o_parent]  
)  
⇒ l_result/nil
```

Description

Convenience function for `axlDBCreatePath`. In one call, without requiring you to create the intermediate path structure, `axlDBCreateLine` creates a path of segments from the list of coordinate points in `l_points`. `axlDBCreateLine` creates `<n-1>` segments from the `<n>` points in `l_points`. All segments are straight and have the width specified by `f_width`. Default width is 0.

If `t_netname` is non-`nil`, the function attaches the line to that net. If `t_layer` specifies a nonetch layer or if the net does not exist an error occurs.

All points are in absolute user units.

Arguments

<code>l_points</code>	List of the vertices (at least two) for this path.
<code>f_width</code>	Width for all segments in the path. Default is 0.
<code>t_layer</code>	Layer to which to add the path. Default is the current active layer.
<code>t_netname</code>	Name of the net to which the path is to belong. The argument may be nonnull only if the path is on an etch layer.
<code>o_parent</code>	<code>dbid</code> of the object to which to attach the path. Use the symbol <code>instance dbid</code> or use <code>nil</code> to specify the design itself.

Value Returned

<code>l_result</code>	List:
-----------------------	-------

(car) list of *dbids* of all paths created or modified

(cadr) t if DRCs are created. Otherwise the function returns nil.

nil Nothing is created.

Example

```
axlDBCreateLine( (list 1000:1250 2000:2250), 15, "etch/top")
⇒ ((dbid:122784) t)
```

This example creates a line at width 15 mils from (1000, 1250) to (2000, 2250) on "etch/top". The command returns the *dbid* of the line and t, indicating that it created DRCs.

axlDBCreateCircle

```
axlDBCreateCircle(  
    l_location  
    [f_width]  
    [t_layer]  
    [o_parent]  
)  
⇒ l_result/nil
```

Description

Convenience function for `axlDBCreatePath`. In one call, creates a circle on the specified layer without requiring you to create the intermediate path construction. The `r_path` of the circle is a single arc with start and endpoints equal.

Arguments

<code>l_location</code>	Center and radius as (X:Y R).
<code>f_width</code>	Width of circle edge. Default = 0.
<code>t_layer</code>	Layer. Default is the active layer.
<code>o_parent</code>	<code>dbid</code> of object to add circle to (symbol instance or <code>nil</code> for design).

Value Returned

<code>l_result</code>	List: (<code>car</code>) list of circle <code>dbids</code> . Normally returns one, however, circles added as etch might touch other etch and break, thus causing more than one <code>dbid</code> . (<code>cadr</code>) <code>t</code> if any DRCs are created. <code>nil</code> if no DRCs are created. <code>nil</code> Nothing was created.
-----------------------	--

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

Example

```
axlDBCreateCircle( (list 1000:1250 1250), 50, "etch/top")
⇒((dbid:122784) t)
```

Creates a circle at location 1000:1250, with a radius of 1250, and a line width of 50, on the "etch/top" layer. The command returns the *dbid* of the line and *t*, indicating that it created DRCs.

Create Shape Interface

You can create shapes using AXL functions as follows:

- To create a simple shape, filled or unfilled, without any voids, first create its boundary path using the `axlPath` functions described earlier. Next, call `axlDBCreateShape` using the path as an argument. `axlDBCreateShape` creates the shape in the database and returns, completing the process.
- To create a shape with voids, first create a shape in “open state” using `axlDBCreateOpenShape`. Next, add voids to the shape as needed using `axlDBCreateVoid` and `axlDBCreateVoidCircle`. Finally, put the shape permanently into the database with `axlDBCreateCloseShape`.

This final function changes the state of the shape from “open” to “closed,” making it a permanent part of the database. Only one shape can be in the “open” state at one time.

You specify both shape and void boundaries with the `r_path` argument, just as you do creating lines and connect lines. `axlDBCreateShape` and `axlDBCreateOpenShape` also check that the following are true:

- All boundary path arguments—shape or void—are closed (equal `startPoint` `endPoint`)
- No boundary path segments touch or cross (no “bow ties”).
- All void boundaries are completely within the boundary of their parent shape

If you fail to meet one or more of these conditions, the functions do not create the shape or void, and return `nil`.

Example

- Closes the shape so that it fills and the command does DRC

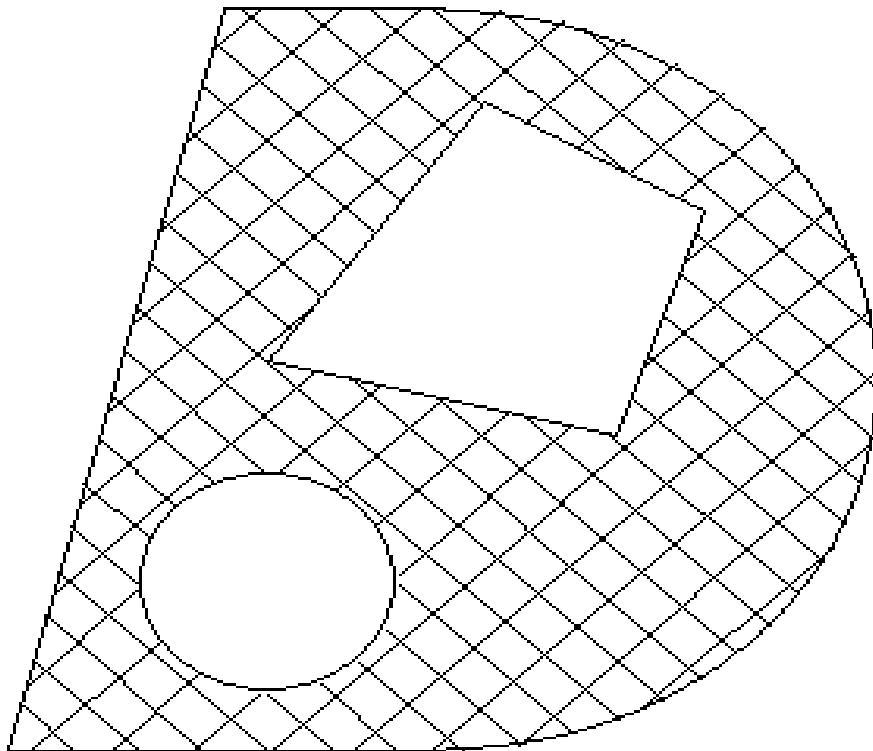
```
mypath = axlPathStart( list(1000:1250))
mypath = axlPathLine( mypath, 0.0, 2000:1250)
mypath = axlPathArcCenter(
    mypath, 0.0, 2000:3250, nil, 2000:2250)
mypath = axlPathLine( mypath, 0.0, 1500:3250)
mypath = axlPathLine( mypath, 0.0, 1000:1250)
myfill1 = make_axlFill( ?angle 45.0, ?origin 10:20,
    ?width 50, ?spacing 80)
myfill2 = make_axlFill( ?angle 135.0, ?origin 10:20,
    ?width 5, ?spacing 100)
myfill = list( myfill1 myfill2)
myshape = axlDBCreateOpenShape( mypath, myfill,
    "etch/top", "sclk1")
if( myshape == axlDBActiveShape()
    println( "myshape is the active shape"))
axlDBCreateVoidCircle( myshape, list(1600:1700 300))
```

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

```
myvoidpath = axlPathStart( list(1600:2300))
myvoidpath = axlPathLine( myvoidpath, 0.0, 2400:2100)
myvoidpath = axlPathLine( myvoidpath, 0.0, 2600:2700)
myvoidpath = axlPathLine( myvoidpath, 0.0, 2100:3000)
myvoidpath = axlPathLine( myvoidpath, 0.0, 1600:2300)
axlDBCreateVoid(myshape, myvoidpath)
axlDBCreateCloseShape( myshape)
```

- Creates a closed path
- Creates the fill structures specifying the crosshatch parameters
- Creates the open shape on "etch/top" associated with net "sclk1" with axlDBCreateOpenShape
- Checks that the shape created is the active shape using axlDBActiveShape which will print the message
- Creates a circular void and attach it to the shape
- Creates a void shape and attach it to the shape



axlDBCreateOpenShape

```
axlDBCreateOpenShape(  
    o_polygon/r_path  
    [l_r_fill]  
    [t_layer]  
    [t_netName]  
    [o_parent]  
)  
⇒ o_shape/nil
```

Description

Creates a shape with the *r_path* boundary, fill parameters, layer, netname, and parent you specify. Returns the *dbid* of the shape in open state. Open state means you can add and delete voids of the shape.

With *o_polygon*, the shape boundary derives from the boundary of the polygon. The holes in the polygon are added as voids to the shape. PCB Editor, APD does not allow hole polygons as input. See `axlPolyFromDB`.

Note: PCB Editor allows only one shape to be in open state at one time.

`axlDBCreateOpenShape` returns `nil` (error) if *l_r_fill* specifies fill on a *t_layer* not allowing filled shapes or unfilled on a *t_layer* requiring filled shapes.

Arguments

o_polygon/r_path Existing path structure created by `axlPath` functions or an *o_polygon* from `axlPolyXXX` interfaces.

l_r_fill One of three possible values:

`t` → create shape solid filled

`nil` → create shape unfilled

List of structures specifying crosshatch parameters for creating the shape:

```
(defstruct axlFill  ;(r_fill) - shape crosshatch data  
  angle          ;angle of the parallel lines  
  origin         ;a point anywhere on any xhatch line  
  width          ;width in user units  
  spacing        ;spacing in user units
```

<i>t_layer</i>	Layer on which to create the shape. Default is the active layer.
<i>t_netname</i>	Name of net to which the shape is to belong. Only allowed on etch layers.
<i>o_parent</i>	<i>dbid</i> of the object to be the parent of the shape. Use the symbol <i>instance</i> or use <i>nil</i> to specify the design itself.

Value Returned

<i>o_shape</i>	<i>dbid</i> of the open shape created. AXL-SKILL does not perform DRC on the shape until you close it using <i>axlDBCreateCloseShape</i> .
<i>nil</i>	No shape created.

Note

An open shape can have voids added to it. It is not DRC checked or filled, until *axlDBCreateCloseShape* is called.

A path starts at *startPoint* and a segment is created for each segment in the *pathList*. If the path does not end at the *startPoint*, it is considered an error. A list of *o_polygons* is not considered valid input. Only a single *o_polygon* is correct input. All path segment coordinates are absolute. PCB Editor, APD allows only one shape to be in open state at one time.

Example 1

```
path = axlPathStart( list( 0:0 400:000 600:400 400:600 0:0))

shp = axlDBCreateOpenShape(path);defaults to a solid filled shape
                                ; unless layer allows unfilled only
; This is optional unless you are adding a shape to etch
; If you do axlDBCreateShape it automatically closes it for you
axlDBCreateCloseShape(car(shp))
```

Example 2

```
p1 = axlPolyFromDB(inElem)  
      ; add it as a unfilled shape on BOARD GEOMETRY/OUTLINE  
res = axlDBCreateShape( car(p1) nil "BOARD GEOMETRY/OUTLINE" )
```

axlDBCreateCloseShape

```
axlDBCreateCloseShape(  
    o_shape  
)  
⇒ l_result/nil
```

Description

Closes the open shape and applies the fill pattern specified in axlDBCreateOpenShape. Then performs DRC. If the fill fails, returns nil.

Arguments

o_shape *dbid* of the open shape created by axlDBCreateOpenShape.

Value Returned

l_result List:

(car) *dbid* of the shape created. This *dbid* is not necessarily the same as the *rd_shape dbid* input argument.

(cadr) t if DRCs are created. nil if DRCs are not created.

nil Nothing was created.

Example

See [Create Shape Interface](#) on page 123 for an example.

axlDBActiveShape

```
axlDBActiveShape(  
)  
⇒ o_shape/nil
```

Description

Returns the *dbid* of the open shape, if any.

Arguments

None.

Value Returned

<i>o_shape</i>	<i>dbid</i> of the active shape created by axlDBCreateOpenShape.
nil	There is no active shape.

Example

See [Create Shape Interface](#) on page 123 for an example.

axlDBCreateVoidCircle

```
axlDBCreateVoidCircle(  
    o_shape  
    l_location  
    [f_width]  
)  
⇒ o_polygon/nil
```

Description

Creates a circular void in the open shape *o_shape*. Calling this function without an open shape causes an error.

Arguments

<i>o_shape</i>	<i>dbid</i> of the open shape created by axlDBCreateOpenShape.
<i>l_location</i>	Center and radius of the circular void to create. The structure of the argument is: (X:Y R).
<i>f_width</i>	Void edge width used by cross-hatch. Default is 0.

Value Returned

<i>o_polygon</i>	<i>dbid</i> of the circular void created.
<i>nil</i>	Error due to calling the function without an open shape. No void is created.

Example

See [Create Shape Interface](#) on page 123 for an example.

axlDBCreateVoid

```
axlDBCreateVoid(  
    o_shape  
    r_path  
)  
⇒ o_polygon/nil
```

Description

Adds a void polygon to the open shape.

Note: It is an error to call this function without an open shape.

Arguments

<i>o_shape</i>	<i>dbid</i> of the open shape created by axlDBCreateOpenShape.
<i>r_path</i>	Existing path structure created by the axlPath functions.

Value Returned

<i>o_polygon</i>	<i>dbid</i> of the void created.
<i>nil</i>	Error due to calling the function with no open shape.

Example

See [Create Shape Interface](#) on page 123 for an example.

axlDBCreateShape

```
axlDBCreateShape(  
    o_polygon/r_path  
    [l_r_fill]  
    [t_layer]  
    [t_netName]  
    [o_parent]  
)  
⇒ l_result/nil
```

Description

Takes the same arguments as `axlDBCreateOpenShape` and adds the `r_path` shape to the database. The difference is that this function creates the shape and puts it into the closed state immediately, rather than leaving it open for modification. Use `axlDBCreateShape` to add shapes without voids.

`axlDBCreateShape` has the same argument restrictions as `axlDBCreateOpenShape`.

Arguments

`o_polygon/r_path` Existing path structure created by `axlPath` functions.

`l_r_fill` One of three possible values:

`t` → create shape solid filled

`nil` → create shape unfilled

List of structures specifying crosshatch parameters for creating the shape:

```
(defstruct axlFill  
    origin  
    width  
    spacing  
    angle)  
; (r_fill) - shape crosshatch data  
; a point anywhere on any xhatch line  
; width in user units  
; spacing in user units  
; angle of the parallel lines
```

Note: As with all SKILL defstructures, use the constructor function `make_axlFill` to create instances of `axlFill`. Use the copy function `copy_axlFill` to copy instances of `axlFill`.

`t_layer` Layer on which to create the shape.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

<i>t_netName</i>	Name of the net to which the shape is to belong.
<i>o_parent</i>	<i>dbid</i> of the object to be the parent of the shape. The parent is a symbol instance or is <i>nil</i> if the design itself.

Value Returned

<i>l_result</i>	List: (car) <i>dbid</i> of the shape created (cadr) <i>t</i> if DRCs are created. <i>nil</i> if DRCs are not created.
<i>nil</i>	Nothing is created.

Example

See [Create Shape Interface](#) on page 123 for an example.

axlDBCreateRectangle

```
axlDBCreateRectangle(  
    l_bBox  
    [g_fill]  
    [t_layer]  
    [t_netname]  
    [o_parent]  
)  
⇒ l_result/nil
```

Description

Creates a rectangle with coordinates specified by *l_bBox*. If the rectangle is not created, the function returns *nil*.

If *t_netname* is non-null, the rectangle becomes a member of that net. Ignores *t_netname* if the rectangle is unfilled.

Does not create the rectangle and returns *nil* (error) in these instances:

- Net does not exist.
- Attempt to create a filled rectangle on an PCB Editor layer requiring an unfilled rectangle.
- Attempt to create an unfilled rectangle on an PCB Editor layer requiring a filled rectangle.

Arguments

l_bBox Bounding box of the rectangle: Lower left and upper right corners of rectangle

g_fill If *t* then the fill is solid. If *nil* (default) then the rectangle is unfilled.

t_layer Layer to which to add the rectangle. Default is the active layer.

t_netname Name of net to which the rectangle is to belong. This argument is meaningful only if the rectangle is being added on an Etch layer.

o_parent *dbid* of object of which the rectangle is to be a part. Use either the *dbid* of a symbol instance or use *nil* to specify the design itself.

Value Returned

<i>l_result</i>	List: (car) rectangle <i>dbid</i> (cadr) t if DRCs are created. nil if DRCs are not created.
nil	Nothing is created.

Example

```
axlDBCreateRectangle(list( 100:100 200:200 ))
  =>(dbid:435623 nil)
(axlDBCreateRectangle '(200:200 400:300) t)

(axlDBCreateRectangle (axlEnterBox) t "ETCH/TOP" 45.0 "net_1")
```

Creates an unfilled rectangle on the active layer with a bounding box of (100:100 200:200) and returns the rectangle's *dbid* in the return list.

Nonpath DBCreate Functions

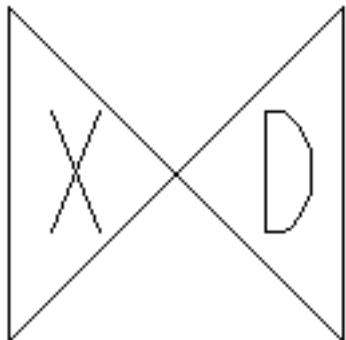
This section describes the DBCreate functions that add nonpath figures to the PCB Editor database.

axlDBCreateExternalDRC

```
axlDBCreateExternalDRC(  
    t_constraint  
    l_anchor_point  
    [t_layer]  
    [lo_dbid]  
    [l_secondPoint]  
    [t_actualValue]  
)  
⇒ l_result/nil
```

Description

Creates an externally-defined (by user) DRC containing the values given in the arguments. An externally defined DRC marker always has the two characters “X D” in it.



The *t_actualValue* argument is optional and provides an externally-defined actual value for the DRC with units.

Attempting to create a DRC object on a non-DRC class is an error.

Note: You can use this function in the layout editor, but not in the symbol editor.

Arguments

<i>t_constraint</i>	Name of the violated constraint. String contains the type of constraint and the required value and comparison.
<i>l_anchor_point</i>	Coordinate of the DRC marker.
<i>t_layer</i>	Layer of the DRC marker.
<i>lo_dbid</i>	Optional list of the objects that caused the DRC (maximum of two).
<i>l_secondPoint</i>	Second reference point. This is a coordinate on the object of the DRC pair that does not have the DRC marker on it. Using this point, you can identify the second object involved in causing the DRC by reading the DRC data in later processes.
<i>t_actualValue</i>	Actual value that caused the DRC.

Value Returned

<i>l_result</i>	List: (car) <i>dbid</i> of the DRC created (always only one) (cadr) t (always)
<i>nil</i>	Nothing is created.

Example

```
axlDBCreateExternalDRC( "Line to Pin--MY SPACING RULE/
    req:12; actual:10", 1500:1800
    "drc error/all", nil, nil, "10 MILS" )
```

Creates a user defined DRC marker at x1500, y1800 to mark a violation of user rule:
"Line to Pin--MY SPACING RULE/req:12; actual:10".

The function adds an "X D" DRC marker at x1500, y1800.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

The DRC marker displays the following information with the *Show – Element* command:

```
LISTING: 1 element(s)
< DRC ERROR >
Class: DRC ERROR CLASS
Subclass: ALL
Origin xy: (1500,1800)
CONSTRAINT: Externally Determined Violation
CONSTRAINT SET: NONE
CONSTRAINT TYPE: LAYOUT
Constraint value: 0 MIL
Actual value: 10 MILS
Properties attached to drc error
EXTERNAL_VIOLATION_DESCRIPTION = Line to Pin--MY SPACING
RULE/req:12; actual:10
-----
```

axlDBCreatePadStack

```
axlDBCreatePadStack(  
    t_name  
    r_drill  
    l_pad  
    [g_nocheck]  
)  
⇒ l_result/nil
```

Description

Adds a padstack *t_name*, using drill hole *r_drill* and pad definition *l_pad*.

Note: This function is not available in the symbol editor.

Defstructs used to create padstack

Drill (r_drill) use make_axlPadStackPad. Elements are:

fixed	= (t/nil) internal fixed flag
drillDiameter	= (float) drill hole size setting
figure	= (symbol) the drill figure. Allowed symbols are NULL, CIRCLE, SQUARE, HEXAGON, HEXAGON_X, HEXAGON_Y, OCTAGON, CROSS, DIAMOND, TRIANGLE, OBLONG_X, OBLONG_Y, RECTANGLE. Note nil is treated as NULL.
figureSize	= ((f_width f_height)) size of drill figure
offset	= ((f_x f_y)) drill hole offset
plating	= (symbol) plate status of drill hole Symbols are: NON_PLATED, OPTIONAL, nil
drillChars	= (string) drill characters identifier. Up to two characters are allowed.
multiDrillData	= list for multiple drill data which is: ((x_num_rows nx_um_columns f_clearance_x [f_clearance_y ["staggered"]])) data type is (int int float [float])
holeType	= (symbol) the hole type. Allowed symbols are CIRCLE_DRILL, OVAL_SLOT, RECTANGLE_SLOT. nil is treated as CIRCLE_DRILL.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

slotSize = ((f_width f_height)) size of slot hole
holeTolerance = ((f_pos f_neg)) +/- hole tolerance
drillNonStandard = (symbol) non-standard drill hole. Allowed symbols are LASER_DRILL, PLASMA_DRILL, PUNCH_DRILL, OTHER_DRILL.

Pad (*l_pad*) structure (up to 3 for each layer)

layer = (string) etch layer name (e.g. "TOP") or "DEFAULT_INTERNAL" if you want one pad layer to map to all internal layers between the top and bottom of the padstack.
type = (symbol) pad type
Allowed symbols: ANTIPAD, THERMAL or REGULAR.
nil is treated as REGULAR.
flash = (string) the pad aperture flash name.
Reference a flash, shape symbol name or nil
for no flash. If this is used for REGULAR pad type then you can only reference a shape symbol (fsm database type).
figure = (symbol) the pad figure
Allowed symbols: NULL, CIRCLE, SQUARE, RECTANGLE, OBLONG_X, OBLONG_Y, OCTAGON. If NULL we check the figureSize and automatically assign a figure type. If you assign a type the figureSize must match that type. For example, a SQUARE must have both width and height of the same value.
figureSize = ((f_width f_height)) height and width of the figure. For a circle other can be 0.
offset = ((f_x f_y)) offset from the padstack origin

Arguments

t_name Padstack name.

r_drill Drill hole data for the padstack.

Note: As with all SKILL defstructs, use the constructor function make_axlPadStackDrill to create instances of

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

`axlPadStackDrill`. See [Create Shape Interface](#) on page 123 for an example.

`l_pad` Pad definition data for the padstack.

Note: As with all SKILL defstructs, use the constructor function `make_axlPadStackPad` to create instances of `axlPadStackPad`. See [Create Shape Interface](#) on page 123 for an example.

`g_nocheck` Optional. `t` disables checks of the padstack definition.
`nil` executes the following checks of the padstack definition:

- Contiguous pad definitions
- Anti-pad / thermal-relief pad definitions
- Existence of two pads with a drilled hole
- A drilled hole with the existence of two pads

Value Returned

`l_result` `dbid` of the padstack created.

`nil` Nothing is created.

Examples

```
; Surface Mount Padstack Example
; See <cdsroot>/share/pcb/examples/skill/dbcreate/pad.il
; The following example adds a surface mount padstack having a
; 25 by 60 rectangular pad on the top layer.
```

```
pad_list = cons(make_axlPadStackPad(
    ?layer "TOP", ?type 'REGULAR,
    ?figure 'RECTANGLE, ?figureSize 25:60) nil)
ps_id = axlDBCreatePadStack("smt_pad", nil, pad_list t)
```

```
; The following example adds a padstack with an 80 diameter circle
; pad on the top layer, 75 diameter circle pad on internal layers,
; 80 square pad on the bottom layer and a 42
; plated thru hole. The drill symbol will be a 60 square.
```

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

```
drill_data = make_axlPadStackDrill(?drillDiameter 42
                                    ?figure 'SQUARE, ?figureSize 60:60, ?plating 'PLATED)
pad_list = cons(make_axlPadStackPad(?layer "TOP",
                                    ?type 'REGULAR, ?figure 'CIRCLE,
                                    ?figureSize 80:80) pad_list)
pad_list = cons(make_axlPadStackPad(
                    ?layer "DEFAULT INTERNAL", ?type 'REGULAR,
                    ?figure 'CIRCLE, ?figureSize 75:75) pad_list)
pad_list = cons(make_axlPadStackPad(
                    ?layer "BOTTOM", ?type 'REGULAR,
                    ?figure 'SQUARE, ?figureSize 80:80) pad_list)
ps_id = axlDBCreatePadStack("thru_pad", drill_data, pad_list t)
```

axlDBCreatePin

```
axlDBCreatePin(  
    t_padstack  
    l_anchorPoint  
    r_pinText  
    [f_rotation]  
)  
⇒ l_result/nil
```

Description

Adds a pin with padstack *t_padstack*, pin name *r_pinText* at location *l_anchorPoint*, and rotated by *f_rotation* degrees.

Note: Use `axlDBCreatePin` only in package and mechanical symbol drawings. Creating a pin in any other type of drawing causes errors.

Arguments

t_padstack Padstack name. If a padstack definition with this name is not already in the layout, the function searches in order the libraries specified by `PADPATH` and loads the definition into the database.

l_anchorPoint Layout coordinates of the location to add the pin.

r_pinText Pin number text structure:

```
(defstruct axlPinText ;(r_pinText) - pin number text data  
    number      ;pin number as a text string  
    offset      ;offset (X:Y) for pin number text  
    text)       ;axlTextOrientation - ;for positioning text
```

This requires the `axlTextOrientation` structure:

```
defstruct axlTextOrientation  
    ;;(r_textOrientation) - description of  
    ;; the orientation of text  
    textBlock    ;string - text block name  
    rotation    ;rotation in floatnum degrees  
    mirrored    ;t-->mirrored, nil --> not mirrored  
    justify)   ;"left", "center", "right"
```

Note: As with all SKILL defstructs, use constructor functions `make_axlPinText` to create instances of `axlPinText` and `make_axlTextOrientation` for `axlTextOrientation`. See [Create Shape Interface](#) on page 123 for an example. Use

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

copy functions `copy_axlPinText` to copy instances of `axlPinText` and `copy_axlTextOrientation` for `axlTextOrientation`.

f_rotation Rotation of pin in degrees.

Value Returned

l_result List:

 (*car*) *dbid* of the pin

 (*cadr*) *t* if DRCs are created. *nil* if DRCs are not created.

nil Nothing is created.

Example

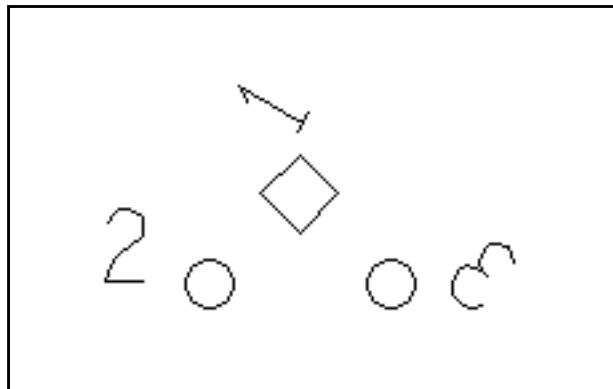
```
mytext1 = make_axlTextOrientation(
    ?textBlock "6", ?rotation 60.,
    ?mirrored nil, ?justify "center")
mypin1 = make_axlPinText(?number "1",
    ?offset 0:75, ?text mytext1)
axlDBCreatePin( "pad1", 0:0, mypin1, 45.)
mytext2 = make_axlTextOrientation(
    ?textBlock "6", ?justify "left")
mypin2 = make_axlPinText( ?number "2",
    ?offset -125:0, ?text mytext2)
axlDBCreatePin( "pad0", -100:-100, mypin2)
mytext3 = make_axlTextOrientation(
    ?textBlock "6", ?rotation -45.,
    ?mirrored t, ?justify "right")
mypin3 = make_axlPinText( ?number "3",
    ?offset 50:0, ?text mytext3)
axlDBCreatePin( "pad0", 100:-100, mypin3)
```

Adds pins "1", "2", and "3" to a package symbol drawing. Pin "1" with a square pad rotated 45 degrees, pins "2" and "3" with round pads, and pin "3" with its pin text mirrored.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

Adds the three pins in the positions shown:



axlDBCreateSymbol

```
axlDBCreateSymbol(  
    t_refdes  
    l_anchorPoint  
    [g_mirror]  
    [f_rotation]  
)  
⇒ l_result/nil  
  
or  
  
axlDBCreateSymbol(  
    l_symbolData  
    l_anchorPoint  
    [g_mirror]  
    [f_rotation]  
)  
⇒ l_result/nil
```

Description

Creates a symbol instance at location *l_anchor_point* with the given mirror and rotation. Examines its first argument to determine what symbol to add, as explained later. Next, searches for the symbol in the symbol definitions, first in the layout, then in the `PSMPATH`. Loads the definition if it is not already in the layout and creates the symbol instance. Returns `nil` a symbol definition is not found.

Note: Do not use this function in the symbol editor.

Arguments

The first argument can be either *t_refdes* or *l_symbolData*, as described here:

t_refdes

If this is the first argument, the function looks for a component in the layout with that refdes, finds the package symbol required for its component device type, adds a package symbol with the symbol name prescribed by the component definition, and assigns that refdes to the symbol (example, refdes U1 requires a DIP14 package symbol). Returns `nil` if it cannot find the given refdes.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

l_symbolData

If this is the first argument, the function looks for the symbol, symbol type, and refdes specified by this structure.

l_symbolData is a list (*t_symbolName* [[*t_symbolType* [*t_refdes*]])], where:

t_symbolName is the name of the symbol (example: DIP14)

t_symbolType is a symbol type: "package" (default), "mechanical", or "format"

t_refdes is an optional refdes; if *t_refdes* is present, *t_symbolType* must be "package".

An example is the list: ("DIP16" "package" "U6")

To create a component with an alternate symbol, that is, a symbol different from the one specified in the component library, use the *l_symbolData* structure. For example, refdes C7 might be a capacitor requiring the top-mount package "CAP1206F". However, your design requires the alternative package "CAP1206B" on the bottom side of the layout.

To create the component mirrored, use *axlDBCreateSymbol* with the *l_symbolData* argument:

"CAP1206B" "package" "C7")

t_refdes

Reference designator of the component associated with the symbol to be created.

l_symbolData

List (*t_symbolName* [[*t_symbolType* [*t_refdes*]])]. (See example above.)

l_anchorPoint

Layout coordinates specifying where to create the symbol.

g_mirror

t → create symbol mirrored

nil → create unmirrored.

f_rotation

Rotation of the symbol in degrees.

Value Returned

axlDBCreateSymbol List:

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

(car) *axl dbid* of the symbol created

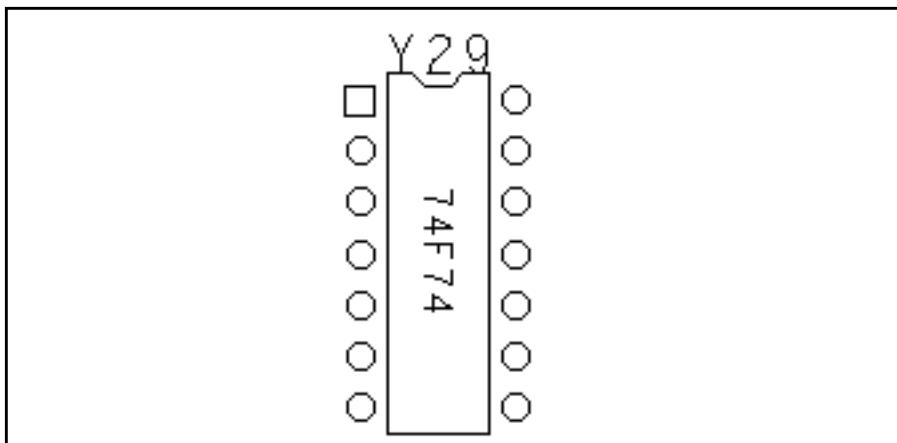
(cadr) t if DRCs are created. nil if DRCs are not created.

nil Nothing is created.

Example

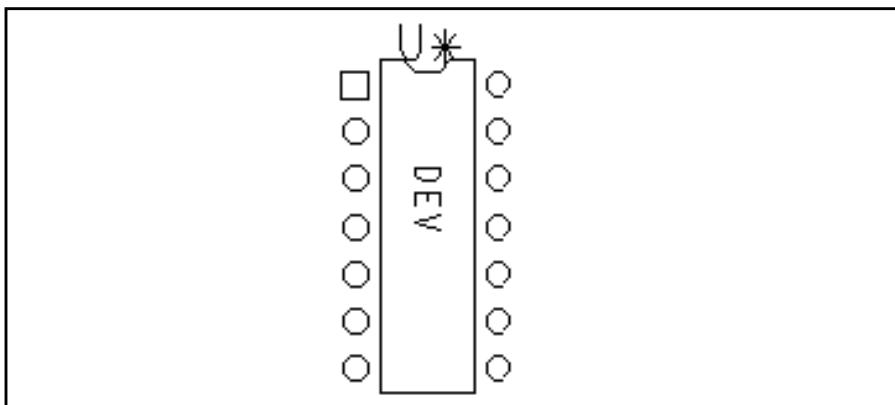
```
axlDBCreateSymbol( "y29", 5600:4600)
⇒(dbid:423143 nil)
```

Creates a symbol with the assigned refdes.



```
axlDBCreateSymbol( list( "dip14" "package" ), 5600:4600)
⇒(dbid:423144 nil)
```

Creates a symbol with the unassigned refdes, just the generic U*:



axlDBCreateSymbolSkeleton

```
axlDBCreateSymbolSkeleton(  
    t_refdes  
    l_anchorPoint  
    g_mirror  
    f_rotation  
    l_pinData  
)  
⇒ l_result/nil
```

or

```
axlDBCreateSymbolSkeleton(  
    l_symbolData  
    l_anchorPoint  
    g_mirror  
    f_rotation  
    l_pinData  
)  
⇒ l_result/nil
```

Description

Creates a “minimal” symbol instance at *l_anchorPoint* with mirror and rotation given but no data in the instance, except the pin data given by *l_pinData*. This is a list of *axlPinData* defstructs defining the data for all pins. The pin count and pin numbers must match that of the library symbol definition. The symbol definition must exist in the database or on LIBPATH.

Behaves like *axlDBCreateSymbol*, except that it adds no symbol data except the symbol pins in the instance. Use to create the “foundation” of a symbol. Then build, using *axlDBCreate* functions to add lines, shapes, polygons, and text as required.

Use, for example, to construct symbols when translated from other CAD systems that define symbol instances in different ways than PCB Editor.

AXL-SKILL applies each *axlPinData* instance in *l_pinData* only to the pin specified by its *number*. (See the description of the *l_pinData* argument below.) A nil value for *l_pinData* means *axlDBCreateSkeleton* adds the pins as they are in the library definition of the symbol. You can selectively customize none, one, or any number of the pins of the symbol instance you create.

Note: Do not use this function in the symbol editor.



This function is intended for programmers with a high level of knowledge of the PCB Editor database model. It provides a powerful method for creating symbols within PCB Editor. Although you can use this command to create non-conventional symbols, the rest of PCB Editor may not behave as you expect. To ensure a symbol behaves as a conventional symbol, you must ensure that what you create abides by symbol rules. For example, you can create a symbol with no attached graphics. PCB Editor's Find utility will not be able to find it. Another programmer may use this feature to create a temporary symbol instance as a placeholder.

Through interaction, the user changes this symbol into a conventional PCB Editor symbol.

Arguments

The first argument may be either `t_refdes` or `l_symbolData`, as described here:

`t_refdes`

If this is the first argument, the function looks for a component in the layout with that refdes, finds the package symbol required for its component device type, adds a package symbol with the symbol name prescribed by the component definition, and assigns that refdes to the symbol (example, refdes U1 requires a DIP14 package symbol). Returns `nil` if it cannot find the given refdes.

`l_symbolData`

If this is the first argument, the function looks for the symbol, symbol type, and refdes specified by this structure.

`l_symbolData` is a list

`(t_symbolName [[t_symbolType [t_refdes]])`, where:

`t_symbolName` is the name of the symbol (example: DIP14)

`t_symbolType` is a symbol type: "package" (default), "mechanical", or "format"

`t_refdes` is a refdes; if `t_refdes` is present, `t_symbolType` must be "package"

Example of a list: ("DIP16" "package" "U6").

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

To create a component with an alternate symbol, a symbol different from the one specified in the component library, use the *l_symbolData* structure.

For example, refdes C7 is a capacitor requiring the top-mount package "CAP1206F". Your design requires the alternative package "CAP1206B" on the bottom side of the layout.

To create the component mirrored, use `axlDBCreateSymbol` with the *l_symbolData* argument:

```
"CAP1206B" "package" "C7")
```

l_anchorPoint Layout coordinates of the location to create the symbol.

g_mirror t → create symbol mirrored

nil → create unmirrored.

f_rotation Rotation of the symbol in degrees.

l_pinData List of `axlPinData` defstructs for any pins you require to be different from their library definition, as shown below:

```
(defstruct axlPinData;(r_pinData) - pin data
  number          ;pin number as a text string
  padstack        ;padstack for the pin (text string)
  origin          ;relative location (X Y) of the pin
  rotation        ;relative rotation of pin in degrees)
```

Note: As with all SKILL defstructs, use the constructor function `make_axlPinData` to create instances of `axlPinData`. Use the copy function `copy_axlPinData` to copy instances of `axlPinData`.

Value Returned

`axlDBCreateSkeleton` List:

(car) `axl dbid` of the symbol created

(cadr) t if DRCs are created. nil if DRCs are not created.

nil Nothing is created.

Allegro PCB and Package User Guide: SKILL Reference

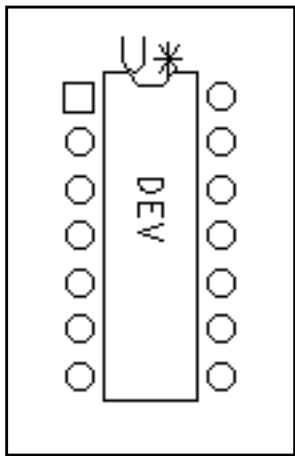
Database Create Functions

Example

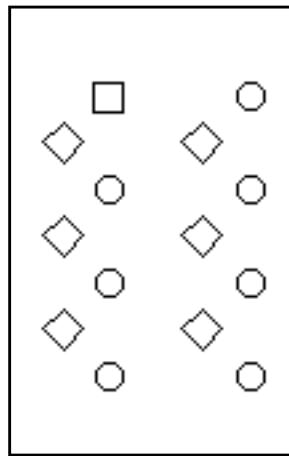
```
mypins = list( make_axlPinData( ?number "2",
    ?padstack "pad1", ?origin -100:-100 ?rotation 45)
make_axlPinData( ?number "4", ?padstack "pad1",
    ?origin -100:-300 ?rotation 45)
make_axlPinData( ?number "6", ?padstack "pad1",
    ?origin -100:-500 ?rotation 45)
make_axlPinData( ?number "9", ?padstack "pad1",
    ?origin 200:-500 ?rotation 45)
make_axlPinData( ?number "11", ?padstack "pad1",
    ?origin 200:-300 ?rotation 45)
make_axlPinData( ?number "13", ?padstack "pad1",
    ?origin 200:-100 ?rotation 45))

axlDBCreateSymbolsSkeleton( list("dip14"),
    5600:4600, nil, 0, mypins)
fi (dbid:426743 nil)
```

Adds a DIP14 symbol with all even-numbered pins having the same padstack as pin 1, rotated 45 °, and offset -100 mils.



Library symbol



Skeleton symbol created
by code sample above

axlDBCreateText

```
axlDBCreateText(  
    t_text  
    l_anchorPoint  
    r_textOrientation  
    [t_layer]  
    [o_attach]  
)  
⇒ l_result/nil
```

Description

Creates a text string in the layout using the arguments described.

Arguments

<i>t_text</i>	Text string to add. axlDBCreateText accepts newlines embedded in the text. Each newline causes the function to create a new text line as a separate database object. The function returns the <i>dbids</i> of all text lines it creates. The <i>textBlock</i> parameter block specified in the <i>axlTextOrientation</i> structure specifies spacing between multiple text lines.
<i>l_anchorPoint</i>	Layout coordinates of the location to add the text.
<i>r_textOrientation</i>	<i>axlTextOrientation</i> structure: <pre>defstruct axlTextOrientation ;;(r_textOrientation) - description of ;; the orientation of text textBlock ;string - text block name rotation ;rotation in floatnum degrees mirrored ;t-->mirrored, nil --> not mirrored justify ;"left", "center", "right"</pre>
Note:	As with all SKILL defstructs, use the constructor function <i>make_axlTextOrientation</i> to create instances of <i>axlTextOrientation</i> . Use the copy function <i>copy_axlTextOrientation</i> to copy instances of <i>axlTextOrientation</i> .
<i>t_layer</i>	Name of the layer on which the text is to be added.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

o_attach *dbid* of the object to which the text must be attached, or use *nil* for the design.

Value Returned

l_result Otherwise the function returns a list:
 (car) list of text *dbids* created, one for each line of text input
 (cadr) *t* if DRCs are created. Otherwise the function returns *nil*.

nil Nothing is created.

Example

```
myorient = make_axlTextOrientation(  
    ?textBlock "8", ?rotation 60.,  
    ?mirrored t, ?justify "center")  
axlDBCreateText( "Chamfer both sides", 7600:4600,  
    myorient, "board geometry/plating_bar", nil)  
⇒(dbid:526743 nil)
```

Adds the text string “Chamfer both sides” center justified, mirrored and rotated 60°.



axlDBCreateVia

```
axlDBCreateVia(  
    t_padstack  
    l_anchorPoint  
    [t_netName]  
    [g_mirror]  
    [f_rotation]  
    [o_parent]  
)  
⇒ l_result/nil
```

Description

Creates a via in the layout as specified by the arguments described below.

Arguments

<i>t_padstack</i>	Padstack name. If a padstack definition with this name is not already in the layout, the function searches in order the libraries specified by <code>PADPATH</code> and loads the definition into the database.
<i>l_anchorPoint</i>	Layout coordinates of the location to create the via.
<i>t_netName</i>	Name of the net to which the via is to belong; <code>nil</code> → via is stand-alone.
<i>g_mirror</i>	<code>t</code> → create via mirrored <code>nil</code> → create via unmirrored.
<i>f_rotation</i>	Rotation of via in degrees.
<i>o_parent</i>	<code>dbid</code> of the object to which to attach the via. Use a symbol instance or use <code>nil</code> to specify the design itself.

Value Returned

<i>l_result</i>	List: (car) <code>dbid</code> of the via created (cadr) <code>t</code> if DRCs are created. <code>nil</code> if DRCs are not created.
-----------------	---

Allegro PCB and Package User Guide: SKILL Reference

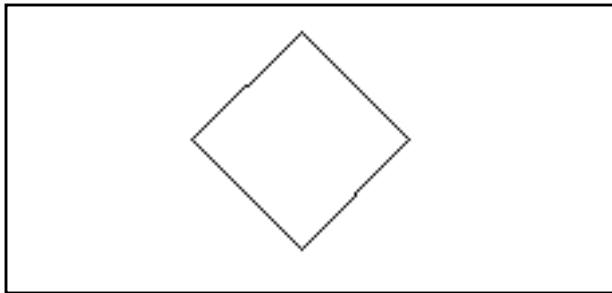
Database Create Functions

nil Nothing is created.

Example

```
myvia = axlDBCreateVia( "pad1", 5600:4200,  
    "sclkl", t, 45., nil)  
    ⇒ (dbid:526745 nil)
```

Adds a standalone via using padstack "pad1" at x5600 y4200 on net "sclk1," mirrored and rotated. Adds a via rotated at 45 degrees:



axlDBCreateSymbolAutosilk

```
axlDBCreateSymbolAutosilk  
  o_symbol  
)  
⇒ t/nil
```

Description

Creates or updates the AUTOSILK information for the specified symbol, as required. Also updates, as required, any other AUTOSILK information near the symbol.

Arguments

o_symbol *dbid* of the symbol.

Value Returned

t A valid symbol *dbid* is provided.

nil The *dbid* provided is not for a valid symbol.

Property Functions

This section describes the `DBCreate` functions you use to create your own (user-defined) property definitions, and add properties to database objects.

axlDBCreatePropDictEntry

```
axlDBCreatePropDictEntry(  
    t_name  
    t_type  
    lt_objects  
    [ln_range]  
    [t_units])  
)  
⇒ o_propDictEntry/nil
```

Description

Creates a property dictionary entry. Use `axlDBCreatePropDictEntry` to add user-defined properties to the property dictionary of your design.

Arguments

t_name Name of the property. Must be different from all other property names in the design, both PCB Editor pre-defined and user-defined property names.

t_type Data type of the property value. The allowed types are:

STRING	LAYER_THICKNESS
INTEGER	IMPEDANCE
REAL	INDUCTANCE
DESIGN_UNITS	PROP_DELAY
BOOLEAN	RESISTANCE
ALTITUDE	TEMPERATURE
CAPACITANCE	THERM_CONDUCTANCE
DISTANCE	THERM_CONDUCTIVITY

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

ELEC_CONDUCTIVITY	VOLTAGE
FAILURE_RATE	VELOCITY

lt_objects List of strings specifying the object types to which this property can be added. To specify only one object type, simply use a string. The allowed object types are:

NETS	COMPONENTS	FUNCTIONS	PINS
VIAS	SHAPES	SYMBOLS	CLINES
LINES	DRCS	FIGURES	DESIGNS
COMPDEFS	PINDEFS	FUNCDEFS	

ln_range List of the lowest and highest legal values for the property. If the first value is *nil*, it means infinitely small. If the second value is *nil*, it means infinity.

t_units String specifying the units of the property value. Supply this argument for a data type (*t_type*) that has no units, such as STRING, INTEGER, or REAL.

Value Returned

o_propDictEntry *dbid* of the property dictionary entry created.

nil Property not created.

Example

```
axlDBCreatePropDictEntry( "myprop", "real", list( "pins" "nets" "symbols"),
list( -50. 100), "level")
propDict:2421543
```

Creates MYPROP as a real number property with range -50 to 100 units of "level", attachable to pins, nets, and symbols.

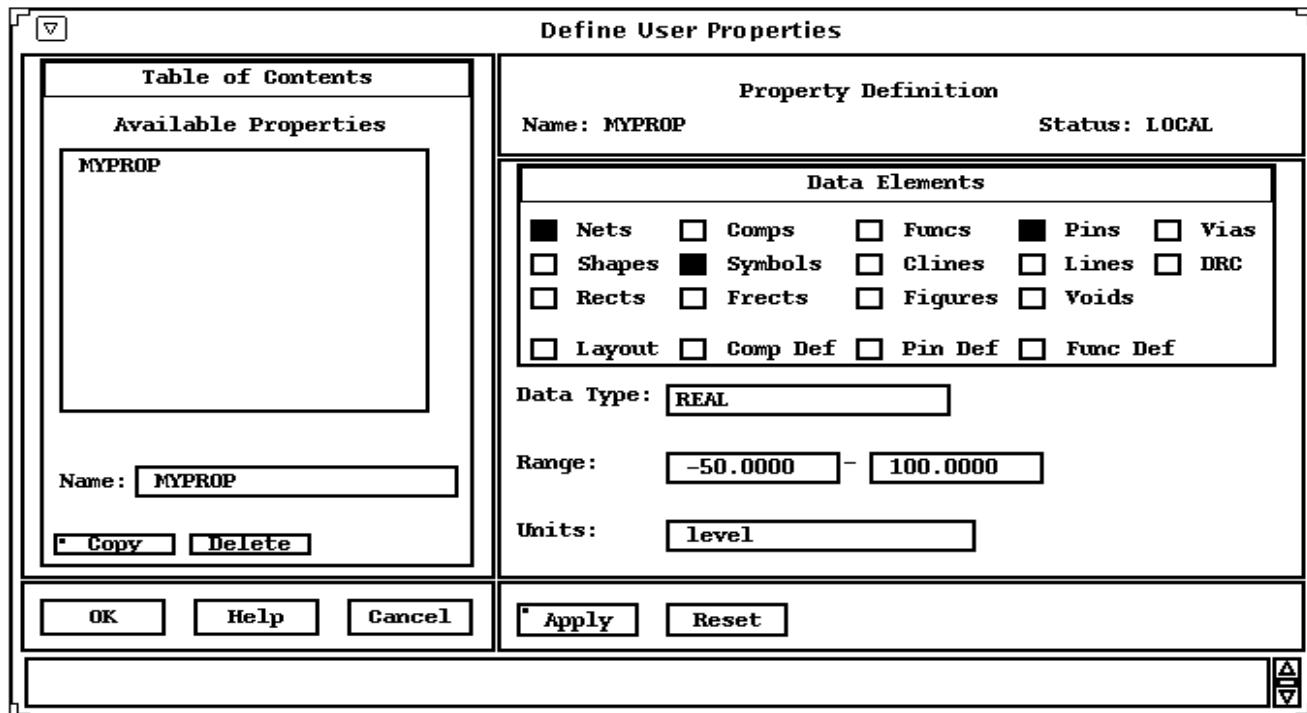
Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

To check

1. From PCB Editor, select *Setup – Property Definitions*.

The Define User Properties window appears.



2. Select *MYPROP* from the Available Properties list.

ax1DBAddProp

```
ax1DBAddProp(  
    lo_attach  
    ll_name_value  
)  
⇒ l_result/nil
```

Description

Adds all the property/value pairs listed in *ll_name_value* to all the object *dbids* listed in *lrd_attach*. If a particular object does not accept a particular property name in *ll_name_value*, `ax1DBAddProp` silently ignores that combination, and continues. If an object already has the specific property attached, `ax1DBAddProp` silently replaces its original value with the one specified in *ll_name_value*.

If any errors occur or if `ax1DBAddProp` has not added or changed any properties, the function returns `nil`.

Arguments

<i>lo_attach</i>	List of PCB Editor object dbids to which to add the property/value combinations listed in <i>ll_name_value</i> . A list of <code>nil</code> denotes attachment to the design (<i>list nil</i>). However, if <i>lo_attach</i> is <code>nil</code> , there are no objects for attachment, and <code>ax1DBAddProp</code> does nothing, returning <code>nil</code> .
<i>ll_name_value</i>	List of property-name/property-value pairs as lists. If the <i>car</i> of this list is not a list, then <code>ax1DBAddProp</code> treats <i>ll_name_value</i> as a single name-value list. The <i>car</i> of each name-value pair is the property name as a string. The <i>cadr</i> of the name-value list is the property value. It is either a string with or without units included, or a simple value (fixed or floating). If the value does not include units explicitly, then <code>ax1DBAddProp</code> uses the units specified in the system <code>units.dat</code> file. <code>ax1DBAddProp</code> ignores the property-value if the property data type is BOOLEAN.

Value Returned

<i>l_result</i>	List: (car) list of <i>dbids</i> of objects that had at least one property successfully added (cadr) always nil.
nil	No properties are added.

Example

```
axlClearSelSet()
axlSetFindFilter(?enabled
    '("NOALL" "ALLTYPES" "NAMEFORM")
    ?onButtons "ALLTYPES") axlSingleSelectName( "NET" "ENA2" )
axlDBAddProp(axlGetSelSet() list( "MYPROP" 23.5))
((dbid:2421543) nil)
```

Attaches the user defined property MYPROP, defined earlier by axlDBCreatPropDictEntry, to net ENA2.

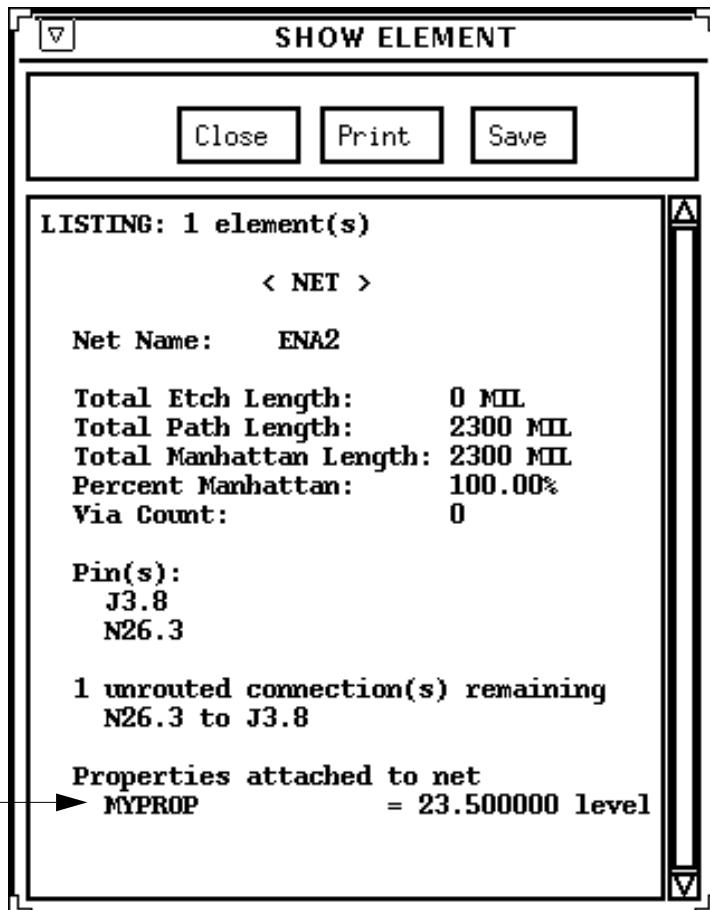
To check

- From PCB Editor, select *Display – Element*, then select any member of net DATA2 to display its properties.

Allegro PCB and Package User Guide: SKILL Reference

Database Create Functions

The Show Element window appears with the MYPROP value at 23.500000 level.



User-defined property
MYPROP attached to
net ENA2

Load Functions

This section describes the *Load* functions that add external objects to the PCB Editor database.

axlLoadPadstack

```
axlLoadPadstack (
    t_padname
)
⇒ o_dbid
```

Description

Loads a padstack by attempting to find the padstack by name in the existing database. Failing that, PCB Editor looks in the pad library on the disk.

Arguments

<i>t_padname</i>	Padstack name. If loaded from disk, PCB Editor uses the PADLIB path variable to find the pad. Padname is limited to 20 characters.
------------------	--

Value Returned

<i>o_dbid</i>	<i>dbid</i> of padstack loaded.
---------------	---------------------------------

nil	Nothing is found.
-----	-------------------

Example

```
pad = axlLoadPadstack (VIA)
```

Loads the VIA padstack.

Parameter Management Functions

Overview

This chapter describes the AXL-SKILL functions that retrieve and set Allegro database parameters. You can access certain Allegro parameters using these functions. Additional functions are built on top of `axlGetParam`/`axlSetParam` to make programming easier.

See [Chapter 1, “Introduction to PCB Editor SKILL Functions.”](#) for a description of available parameter attributes.

The use model follows:

- Get the parameter using `axlGetParam`.
- Modify the values using `axlSetParam`.
- Update the parameter using `axlSetParam`.

AXL-SKILL restricts you from creating new parameters or subclasses.

axlColorGet

```
axlColorGet(  
            x_number/`background  
          ) -> lx_rgb/nil  
axlColorGet(  
            'count  
          ) -> x_count  
axlColorGet(  
            'all  
          ) -> llx_rgb
```

Description

Get color palette. Supports the following modes:

- If passed, an index less the color count returns a list containing the red, green, blue palette values for that color index. These are integer values between 0 (no color) and 255 (maximum color). For example, a value of 255 255 255 is white. Or if passed, '*background*' returns the palette for the background.
- If given '*count*' returns the current size of the database palette (currently always 24).
- If passed '*all*' returns a list of list (red, green, blue) for all entire database palette EXCEPT the background. The color index is the number assigned to each layer in PCB Editor. (see `axlVisibleGet`).

Arguments

<i>x_number</i>	Color number.
' <i>background</i>	Get background color.
' <i>count</i>	Query current database color palette size.
' <i>all</i>	Get entire database color palette (except background).

Value Returned

<i>x_count</i>	Size of database palette.
<i>nil</i>	Error.

lx_rgb A palette.

lx_rgb The entire database palette.

Examples

Get red/green/blue of color 2:

```
clr = axlColorGet(2)
```

Get background color:

```
bground = axlColorGet(`background)
```

Get number of colors:

```
cnt = axlColorGet(`count)
```

Get all red/green/blue color settings except background:

```
all = axlColorGet(`all)
```

axlColorShadowGet

```
axlColorShadowGet(  
    g_option  
) -> t/nil/x_percent
```

Description

Provides the options of shadow mode.

Arguments

g_option

'mode	Shadow mode status (<i>t</i> is on, <i>nil</i> is off).
'activeLayer	Active layer dimming enabled (<i>t</i>).
'percent	Current brightness percentage (0 to 100).

Value Returned

t/nil Shadow or active layer mode on or off.

x_percent Brightness percentage.

See also `axlColorSet`, `axlColorShadowSet`

Examples

Is shadow mode on:

```
axlColorShadowGet( 'mode )
```

Is shadow mode percent:

```
axlColorShadowGet( 'percent )
```

axlColorShadowSet

```
axlColorShadowSet(  
    'mode / 'activeLayer  
    t / nil  
) -> t / nil  
  
axlColorShadowSet(  
    'percent  
    x_percentage  
) -> t / nil
```

Description

Sets the shadow mode options. These are equivalent to the color commands in the shadow mode box under the Display group.

Options are:

- The mode option is either `t` or `nil` to turn shadow mode on or off.
- The activeLayer option is either `t` or `nil` to automatically dim the active layer.
- The percent option sets the dimness (0) to brightness (100) percentage.

Note: On graphics or display combinations, shadow values of less than 40 percent disappear into the background.

After you finish all the color changes, call `axlVisibleUpdate` to update the display.

This interface is disabled if you set the `display_noshadow` environment variable.

Arguments

<code>'mode</code>	Enable or disable shadow mode.
<code>'activeLayer</code>	Enable or disable active layer dimming.
<code>'percent</code>	Set shadow mode percentage (0 to 100)

Value Returned

t If successful.

nil An argument error.

See also `axlColorSet`, `axlColorShadowGet`, `axlVisibleUpdate`

Examples

Is shadow mode on:

```
axlColorShadowSet('mode t')
```

Is shadow mode percent:

```
axlColorShadowSet('percent 20')
```

axlColorLoad

```
axlColorLoad(  
    t_file/nil  
) -> t/nil
```

Description

Loads a PCB Editor color file (default .col file). Master color file is located at <cdsroot>/share pcb/text/lallegro.col.

File format is:

```
#      Comment if in first column.  
#N    Next line with a number is number of colors (currently only 24 is supported).  
This should appear first in the file.  
Number format  
#Number  
24  
#B - next line with a number is background color. This should appear after color  
number. Format of color line must be:  
(name is currently ignored):  
0 <red> <green> <blue> [<name>]  
EXAMPLE of background format setting it to black  
#Background Color  
0      0      0      0  
#I - next set of lines sets the colors. These should always appear last in the file.  
We will read until the first color number that exceeds the color number (currently  
hardcoded as 24) or the end of file is reached. The order the colors appear in the  
file determines the initial color [priority (highest (first) to lowest (last))].  
Format is:  
<color number> <pen number> <red> <green> <blue> [<name>]  
EXAMPLE:  
1      1      255      255      255      White  
2      2      14       210      255      LtBlue  
<color number>: entry in color table. This is the color number referenced by the  
allegro subclass (axlLayerGet)  
<pen number>: Used by Allegro plot (UNIX) to control what pen to use during  
plotting. Not applicable on Windows.  
<red> intensity of red to blend into color 0 to 255  
<green> intensity of green to blend into color 0 to 255  
<blue> intensity of blue to blend into color 0 to 255  
<name> (optional) name of color, currently not used by Allegro but sigxp takes  
advantage of the name to auto-assign colors.
```

Call `axlVisibleUpdate` to update the display after you finish manipulating the colors.

In PCB Editor, you need the color file to start a new design. Opening existing databases uses the color table stored in that database. A new database created, when PCB Editor is already running, copies the color table from the previous database.

Arguments

<code>s_file</code>	Color file name to load.
<code>nil</code>	Uses <code>lallegro.col</code> . If no directory path, PCB Editor uses the <code>LOCALPATH</code> environment variable to find the file.

Value Returned

<code>t</code>	If loaded file.
<code>nil</code>	File not found or error in loading file.

Example

Load user-defined default color. Overriding and setting current board values:

```
axlColorLoad(nil)  
axlVisibleUpdate(t)
```

See also `axlColorSave`, `axlColorSet`.

axIColorOnGet

```
    x_colorNumber
) -> g_state/nil

    'count
) -> x_count

    'all
) -> lg_state

    lx_colorNumber
) -> lg_state
```

Description

Provides access to the color priority command functionality, providing an additional way of controlling the visibility of colors on the drawing canvas.

Modes are:

- If given a color number, returns whether that color is on or off.
- If given the symbol '*count*', returns the number of colors available in the design (twenty-four).
- Returns the visibility state of all colors.
- Given a list of color numbers, returns the visibility of each color.

Arguments

x_colorNumber Color number. Value between one and maximum colors.

lx_colorNumber List of color numbers.

'count Return maximum colors.

'all Returns visibility state of all colors.

Value Returned

<i>g_state</i>	t color is on, nil color is off.
<i>x_count</i>	Total number of colors in design.
<i>lg_state</i>	List of visibility state for all colors.
nil	nil return may indicate an error.

Examples

If color number 2 is on:..

```
clr = axlColorOnGet(2)
```

Get number of colors:

```
cnt = axlColorOnGet(`count)
```

Get all red/green/blue color settings except background:

```
all = axlColorOnGet(`all)
```

Get color numbers

```
state = axlColorOnGet('(1 5 6))
```

See also `axlColorSet`, `axlColorOnSet`.

axlColorOnSet

```
x_colorNumber/lx_colorNumber  
g_state  
) -> t/nil  
  
'all  
g_state  
) -> t/nil
```

Description

Sets the visibility of colors. Modes are:

- Either a single color number or list color numbers can be provided with turn visibility on (t) or off (nil).
- All color numbers can be turned on or off by using the 'all symbol. axlVisibleUpdate should be called when you complete *all color*. You cannot turn the background on or off.

Arguments

x_colorNumber Color number between one and maximum colors.

lx_colorNumber List of color numbers.

g_state t for on and nil for off.

'*all* All color numbers.

Value Returned

t Success.

nil Illegal value provided.

Examples

Set color number three to on:

```
axlColorOnSet( 3 t )
```

Set color number all colors off:

```
axlColorOnSet( 'all nil )
```

Set color number six, seven, eight to on:

```
axlColorOnSet( '( 6 7 8 ) t )
```

See also `axlColorSet`, `axlColorOnGet`, `axlVisibleUpdate`

axIColorPriorityGet

```
'all  
    ) -> lx_colorNumbers  
  
        x_colorNumber  
        ) -> x_priority/nil  
  
    'priority  
        x_priority  
        ) -> x_colorNumber/nil
```

Description

This allows an application to query the color priority mapping for color numbers. Colors with a lower priority number are drawn before colors at higher numbers. This interface provides access to the data displayed by color priority command. In this command's dialog, lower to higher priorities are displayed top to bottom. No two colors may have the same priority.

The modes supported by this function are:

- Returns a list of priorities for all color numbers in the design. The index into the list is the colorNumber while the value is the priority of the color. In the example below, the priority of colorNumber one is 3, the priority of color number two is 1.
- Given a colorNumber returns its priority
- Given a priority, returns the associated colorNumber.

Arguments

<i>'all</i>	Return color priority table.
<i>'priority</i>	Argument is the color number.
<i>x_colorNumber</i>	Obtain priority of given color number.

Allegro PCB and Package User Guide: SKILL Reference

Parameter Management Functions

x_priority Obtain color at given priority.

Value Returned

x_priority Priority of given color.

x_colorNumber Color number at given priority.

lx_colorNumbers Color priority table.

nil Error based on color number or priority number greater than maximum colors.

Examples

Return priority of all colors:

```
axlColorPriorityGet('all)
-> (3 1 4 2 6
 5 8 7 9 10
```

Using example above, return priority of the given color number:

```
axlColorPriorityGet(2)
-> 1
```

Using example above, return the color number at given priority:

```
axlColorPriorityGet('priority 2)
```

See also `axlColorPrioritySet`, `axlColorSet`

axlColorPrioritySet

```
    lx_colornumbers
) -> t/nil

    'top
    x_colorNumber
) -> t/nil

    'change
    x_oldPriority
    x_newPriority
) -> t/nil

    x_colorNumber
    x_newPriority
) -> t/nil
```

Description

This changes the priority associated with a color. Several commands exist.

1. Assigns list of colors a priority. The colorNumber is the index of the list while the value at that index is its priority (See example in `axlColorPriorityGet`).
2. If the same priority appears multiple times in the list then its priority associates with its final appearance. If the list is a partial list then priorities of colorNumbers not appearing in the list may change to insure no two colors have the same priority. You can use the output of `axlColorPriorityGet('all)` with this option.
3. Makes a color number the top priority. This is a version of:
`axlColorPrioritySet('change axlGetPriority(x_colorNumber) 1).` It duplicates the functionality commands used for putting the color of the active layer at the top of the color draw stack.
4. The 'change mode duplicates color priority form functionality. It takes the color number associated with the old priority and moves it to the new priority. All colors between the old and new priority are pushed towards the opening created at the old priority location.
5. Assigns the priority to a color number. Call `axlVisibleUpdate` at the end of color changes to update the display.

Arguments

' <i>all</i>	Returns color priority table.
' <i>priority</i>	Next argument is the color number.
<i>x_colorNumber</i>	Obtain priority of given color number.
<i>x_priority</i>	Obtain color at given priority.

Value Returned

<i>t</i>	Success.
<i>nil</i>	Error in one of the arguments.

Examples

Set priority of all colors:

```
axlColorPrioritySet('all)
```

Set color number two at top priority:

```
axlColorPrioritySet('top 2)
```

Move the color number at priority two to priority five. Color at five moved to four, four to three and three to two:

```
axlColorPrioritySet('change 2 5 )
```

Assign color four priority one:

```
axlColorPrioritySet(4 1 )
```

See also `axlColorPriorityGet`, `axlColorSet`, `axlVisibleUpdate`

axlColorSave

```
axlColorSave(  
    t_file/nil  
) -> t/nil
```

Description

Saves current design colors to specified file.

Argument

t_file File name. If nil; saves to <HOME>/pcbenv/lallegro.col.
If no extension, uses .col extension.

Value Returned

t Successful.

nil Failed to save.

EXAMPLES

Save current design color settings:

```
axlColorSave( "mycolor" )
```

See also [axlColorSave](#), [axlColorSet](#)

axlColorSet

```
axlColorSet(  
    x_number / 'background  
    l_rbg  
) -> t/nil  
  
axlColorSet(  
    'all  
    ll_rgb  
) ->t/nil
```

Description

Sets red, green, blue palette for a color number or background.

Modes supported:

- Color number (*x_number*) and red/green/blue list. *x_number* must be between one and axlColorGet('count), or 'background sets red/green/blue as the background color.
- 'all takes a list of red/green/blue values and sets colors starting at one to the end of the list. Intended to use with axlColorGet('all) to save or restore color values.

Red/green/blue colors are values between 0 (least intesity) to 255 (maximum intesity).

After color changes are made, call axlVisibleUpdate to update the display.

Color model:

A color (or colorNumber) in PCB Editor has the following attributes:

- A palette of red, green and blue values between 0 and 255. 0 adds none of the primary color to the mixture while 255 adds the maximum. For example, 0 , 0 , 0 is black and 255 , 255 , 255 is white. The color mixture is controlled using the palette section of the color command.
- The option of color on or off (this state is not saved with the database) provided by the checkbox in the color priority command.

Allegro PCB and Package User Guide: SKILL Reference

Parameter Management Functions

- Drawing priority where one is the highest priority. No two colors may have the same priority. Use the color boxes in the color priority command to rearrange the colors.
- Each color number can be assigned to a layer. Multiple layers will have the same color number, because there are more layers than colors, .
- PCB Editor supports setting a background palette value. Grids, ratsnest, temporary highlight can have a color number assigned via axlDBControl.

Color services:

axlColorSet	This routine.
axlColorGet	Get red, green, or blue of one or more color numbers.
axlColorOnGet	Get if color number is turned on or off.
axlColorOnSet	Set color number on or off.
axlColorPrioritySet	Control drawing priority of color numbers.
axlColorPriorityGet	Provides the color priority.
axlColorShadowGet	Shadow mode options.
axlColorShadowSet	Set shadow mode options.
axlColorSave	Save color values to file.
axlColorLoad	Load color values from file.
axlUIColorDialog	Standard color chooser dialog box.
axlDBControl	Miscellaneous color number assignments (for example, highlight).
axlLayerGet	Get layer (class/subclass) attributes (control color) number and visibility for individual layers.
axlLayerSet	Set color number or visibility for a layer.
axlVisibleLayer	Set visibility of layer.
axlIsVisibleLayer	Provides the layer visibility.

Allegro PCB and Package User Guide: SKILL Reference

Parameter Management Functions

axlVisibleGet	Get visibility set for design.
axlVisibleSet	Set visibility set for design.
axlVisibleDesign	Global design visibility control.
axlVisibleUpdate	Update windows with color changes.

Arguments

x_number	Color index.
'background	Set background color.
'all	Set colors based upon a list starting at color number one.
l_rgb	Red/green/blue lists; three integers.
ll_rgb	Lists of red/green/blue values.

Value Returned

t	Successful.
nil	An error; wrong arguments: color number is less than one or greater than maximum.

EXAMPLES

Set color number three same as color two:

```
clr = axlColorGet(2)
axlColorSet(3 clr)
axlVisibleUpdate(nil)
```

Set first three colors:

```
axlColorSet('all '((10 10 10) (40 40 40) (100 100 100)))
```

axlGetParam

```
axlGetParam (
    t_parm_name
)
⇒ fio_paramDbid/nil
```

Description

Gets the parameter *dbid* for a named object. For descriptions of supported parameter names, see are [Chapter 2, “The PCB Editor Database User Model.”](#)

Arguments

t_parm_name Name of the parameter to seek. The legal naming conventions follow:

```
paramTextBlock: <#> where # is 1-16; e.g., paramTextBlock:1
paramDesign
paramDisplay
paramLayerGroup:<name> where name is legal Allegro class name
paramLayerGroup:<name>/paramLayer:<name>
```

Value Returned

o_paramDbid *dbid* for the requested parameter.

nil Parameter requested not found.

Example

```
axlGetParam ( "paramTextBlock:5" )
dbid: 541678
```

Returns text block 5.

axlSetParam

```
axlSetParam (
  o_paramDbid
)
⇒ o_paramDbid/nil
```

Description

Takes an existing parameter *paramdbid* and modifies the existing Allegro database. You get this *dbid* as the result of using axlGetParam. Modifies all changed fields in one operation.

Arguments

<i>o_paramDbid</i>	Parameter id returned from axlGetParam. Modifying the contents of a record retrieved from GetParam changes an existing parameter.
--------------------	---

Value Returned

<i>o_paramDbid</i>	Returns the input parameter id if successful
nil	Database was not modified.

Database Layer Management

These functions allow easier access to layer attributes.

axlDBGetLayerType

```
axlDBGetLayerType  
  t_layerName  
  
)  
⇒ t_layertype/nil
```

Description

Retrieves the cross-section type of a given layer. This may be (Layer Type in define xsection form): CONDUCTOR, CROSSOVER, DIELECTRIC, PLANE, BONDING WIRE, MICROWIRE, MULTIWIRE, OPTICAL WAVE GUIDE, or THERMAL GLUE COATING.

Arguments

t_layername Layername is <class>/<subclass>.

Value Returned

t_layertype Layer type string.

nil Layer is invalid.

Example

```
axlDBGetLayerType( "ETCH/TOP" )  => "BONDING_WIRE"
```

axlIsLayer

```
axlIsLayer(  
    t_layer  
)  
⇒ t/nil
```

Description

Determines if the *t_layer* exists. *t_layer* is a fully qualified layer name.

Arguments

<i>t_layer</i>	Name of layer in format “<class>/<subclass>.”
----------------	---

Value Returned

t	Layer exists.
---	---------------

nil	Layer does not exist.
-----	-----------------------

axlIsVisibleLayer

```
axlIsVisibleLayer(  
    t_layer  
)  
⇒ t/nil
```

Description

Returns the visibility (*t/nil*) of a fully qualified layer.

Arguments

t_layer Name of layer in format "*<class>/<subclass>*".

Value Returned

t Layer is visible.

nil Layer is invisible or not present.

Example

```
axlIsVisibleLayer( "pin/top" ) ⇒ t
```

axlLayerCreateCrossSection

```
axlLayerCreateCrossSection(  
    t_Prev_layerName  
    t_layerType  
    t_materialType  
    [t_subclassName]  
    [t_planeType]  
)  
⇒ t/nil
```

Description

Adds a new cross-section layer to the design.

Arguments

<i>t_Prev_layerName</i>	Name of the layer above which the new layer is to be added
<i>t_layerType</i>	Type of layer to be added, such as Conductor or Surface.
<i>t_materialType</i>	Material of the layer.
<i>t_subclassName</i>	Name of the new layer.
<i>t_planeType</i>	Type of plane, either Positive or Negative. The default is Positive.

Value Returned

t	Layer is created or already exists.
nil	Layer does not exist and could not be created.

Example

```
axlLayerCreateCrossSection( "Bottom" "Conductor"
    "Copper" "New_Layer" "Positive" );
```

Creates a conductor layer with these characteristics:

- Located above a layer subclass called Bottom
- Has copper layer material
- Has a subclass name New_Layer
- Positive plane

axlLayerCreateNonConductor

```
axlLayerCreateNonConductor(  
    t_layerName  
)  
⇒ t/nil
```

Description

Creates a new subclass for non-etch subclasses. AXL-SKILL restricts you from creating etch subclasses.

Arguments

t_layerName *<class>/<subclass>*

Value Returned

t	New subclass is created or, subclass already exists.
nil	New subclass is not created.

Example

```
axlLayerCreateNonConductor( "BOARD GEOMETRY/MYSUBCLASS" )
```

Creates a new subclass named MYSUBCLASS.

axlLayerGet

```
axlLayerGet(  
    t_layer  
)  
⇒ o_dbid/nil
```

Description

Gets the layer parameter given the shortcut notation of *<class>/<subclass>*. See also `axlSetParam`.

Arguments

t_layer Name of layer in format "*<class>/<subclass>*".

Value Returned

o_dbid Layer parameter *dbid*.

nil Layer is not present.

Example

Changes color of top etch layer.

```
q = axlLayerGet( "ETCH/TOP" )  
    q->color = 7  
    axlLayerSet(q)  
    axlVisibleUpdate(t)
```

axlVisibleDesign

```
axlVisibleDesign(  
    g_makeVis  
)  
⇒ t/nil
```

Description

Makes entire design visible or invisible. This command does not visually change the display, since it can also be used in conjunction with the `axlSelect` command family to provide additional filtering of the database objects. If you wish to visually update the display, call `axlUIWUpdate(nil)` after changing the visibility.

Note: This routine along with `axlVisibleGet` and `axlVisibleSet` allows you to temporarily change the visibility of the design to provide additional filtering capability when finding objects via the selection set. The programming model is:

```
saveVis = axlVisibleGet()  
axlVisibleDesign(nil)  
; set desired layers visible via one or more calls to  
axlVisibleLayer(...)  
; set find filter for objects to find  
axlSetFindFilter(...)  
; find objects by using one of the Select APIs .. example  
axlAddSelectAll()  
objs = axlGetSelSet()  
  
; restore visibility  
axlVisibleSet(saveVis)  
; note no need to make a call to axlVisibleUpdate because  
; the visibility changes are a wash
```

Arguments

g_makeVis Either `t` or `nil`.

`t` = make entire design visible

`nil` = make entire design invisible

Value Returned

t Design made visible or invisible as specified.

nil Should never be seen.

Note: This command does not visually change the display. To visually update the display, call `axlUIWUpdate (nil)` after changing the visibility.

axlVisibleGet

```
axlVisibleGet()  
)  
⇒ l_visList/nil
```

Description

Returns the visibility of the entire design - which layers are visible/invisible.

Arguments

None.

Value Returned

l_visList List of lists. The format is for each class:

```
(nil class <t_className> visible t/nil/-1  
subclassinfo <l_subclass>)  
....)  
l_subclass format:  
((<t_subclass> t/nil) ....)  
where t/nil/-1  
t - visible  
nil - invisible  
-1 - class has both visible and invisible components.
```

Note: Any change in the structure of *l_vislist* affects axlVisibleSet, this function's complementary function.

Example

```
visList = axlVisibleGet()  
(  
(nil class "BOARD GEOMETRY" visible nil subclassinfo nil)  
(nil class "COMPONENT VALUE" visible nil subclassinfo nil)  
(nil class "DEVICE TYPE" visible nil subclassinfo nil)  
(nil class "DRAWING FORMAT" visible nil subclassinfo nil)  
(nil class "DRC ERROR CLASS" visible t subclassinfo nil)  
(nil class "ETCH" visible -1  
subclassinfo  
(("TOP" t)  
("TRACE_2" nil))
```

Allegro PCB and Package User Guide: SKILL Reference

Parameter Management Functions

```
( "TRACE_3" nil)
( "BOTTOM" t)
))
(nil class "MANUFACTURING" visible nil subclassinfo nil)
(nil class "ANALYSIS" visible nil subclassinfo nil)
(nil class "PACKAGE GEOMETRY" visible nil subclassinfo nil)
(nil class "PACKAGE KEEPIN" visible t subclassinfo nil)
(nil class "PACKAGE KEEPOUT" visible nil subclassinfo nil)
(nil class "PIN" visible t subclassinfo nil)
(nil class "REF DES" visible nil subclassinfo nil)
(nil class "ROUTE KEEPIN" visible t subclassinfo nil)
(nil class "ROUTE KEEPOUT" visible nil subclassinfo nil)
(nil class "TOLERANCE" visible nil subclassinfo nil)
(nil class "USER PART NUMBER" visible nil subclassinfo nil)
(nil class "VIA CLASS" visible nil subclassinfo nil)
(nil class "VIA KEEPOUT" visible nil subclassinfo nil)
)
```

Returns the visibility of the entire design.

axlVisibleLayer

```
axlVisibleLayer(  
    t_layer  
    g_makeVis  
)  
⇒ t/nil
```

Description

Sets a given layer to visible or invisible. If given only a class name, sets the entire layer to visible or invisible.

Arguments

<i>t_layer</i>	Name of the layer. Either a fully qualified layer name in the format <class>/<subclass> or a class name in the format <class>.
<i>g_makeVis</i>	Either <code>t</code> or <code>nil</code> . <code>t</code> = make visible <code>nil</code> = make invisible.

Value Returned

`t` Layer set to visible or invisible as specified.

`nil` Layer does not exist.

Note: This command does not visually change the display. To visually update the display, call `axlUIWUpdate (nil)` after changing the visibility.

axlVisibleSet

```
axlVisibleSet(  
    l_visList  
)  
⇒ t/nil
```

Description

Sets the visibility of the entire design.

Arguments

l_visList List with visibility attributes.

Value Returned

t Set the visibility of the design as specified.

nil Incorrect format for *l_visList*.

Note: This command does not visually change the display. To visually update the display, call `axlUIWUpdate (nil)` after changing the visibility.

axlConductorBottomLayer

```
axlConductorBottomLayer(  
)  
⇒ name
```

Description

Returns the name of the bottom conductor layer.

Arguments

none

Value Returned

Name of the bottom conductor layer.

axlConductorTopLayer

```
axlConductorTopLayer(  
)  
⇒ name
```

Description

Returns the name of the top conductor layer.

Arguments

none

Value Returned

Name of the top conductor layer.

axlDBCreateFilmRec

```
axlDBCreateFilmRec(  
    t_filmname  
    n_rotate_code  
    n_x_offset  
    n_y_offset  
    n_undef_line_width  
    n_shape_bound  
    n_plot_mode  
    n_mirrored  
    n_full_contact  
    n_supp_unconnect  
    n_draw_pad  
    n_aper_rot  
    n_fill_out_shapes  
    n_vector_based  
)  
⇒ t_name/nil
```

Description

Creates a film record with parameters and visible layers specified.

[Example 1](#) on page 204 shows a common use of this function through the film control form's film record *load* option, which calls axlfcreate using the following form:

```
(axlfcreate t_name l_params l_vis)
```

The axlfcreate call includes these arguments and passes all parameters needed as arguments to axlDBCreateFilmRec:

<i>t_name</i>	Name of the record to be created, of the form "NAME"
<i>l_params</i>	List consisting of 13 numbers, of the form '(0 0 0 ...), which correspond to the <i>Film Options</i> fields in the film control form.
<i>l_vis</i>	The list of visible layers, each layer enclosed in quotes, space delimited, of the form '("ETCH/TOP" "VIA/TOP" . . .)

axlfcreate first makes the specified layers visible, then calls axlDBCreateFilmRec, providing the filmname and the 13 numbers it was passed as *l_params*.

Arguments

<i>t_filmname</i>	The name of the film record to create.
<i>n_rotate_code</i>	0, 2, 4, or 6, corresponding to 0, 90, 180, or 270 degree rotation.
<i>n_x_offset</i>	Film record block origin x offset.
<i>n_y_offset</i>	Film record block origin y offset.
<i>n_undef_line_width</i>	Film record undefined line width.
<i>n_shape_bound</i>	Shape bounding box size.
<i>n_plot_mode</i>	Film record plot mode -- 0 = NEGATIVE, 1 = POSITIVE.
<i>n_mirrored</i>	Flag denoting mirroring.
<i>n_supp_unconnect</i>	Indicator to not flash unconnected pads
<i>n_draw_pad</i>	Indicator to draw pads.
<i>n_aper_rot</i>	Boolean indicator for aperture rotation.
<i>n_fill_out_shapes</i>	Boolean indicator to fill outside shapes. This is the opposite of the "suppress shape fill" switch in the film control form, for example, if suppress shape fill is selected, <i>fill_out_shapes</i> should be 0. This is named this way because <i>art_film_block_type</i> structures have <i>fill_out_shapes</i> fields instead of <i>suppress_shape_fill</i> fields.
<i>n_vector_based</i>	Boolean indicator for vector-based pad behavior.

Value Returned

<i>t_name</i>	Name of the film record created.
<i>nil</i>	Failure to create the film record.

Example 1

```
(axlfcreate "TRACE_2" '(0 0 0 0 0 1 0 0 0 0 0 0 1) '("ETCH/TRACE_2" "PIN/TRACE_2"  
"VIA CLASS/TRACE_2" ))
```

- Called through film control form

Call the film control form's film record save option, which creates a .txt file.

axlfcreate changes layer visibility and calls axlDBCreateFilmRecord as shown:

```
(axlDBCreateFilmRecord "TRACE_2" 0 0 0 0 0 1 0 0 0 0 0 0 1)
```

Select the /load option, which evaluates the contents of the file, calling axlfcreate.

Example 2

```
(axlDBCreateFilmRecord "TOP" 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

Creates a film record TOP with current visible layers and fields initialized to 0/false/default:

axlSetPlaneType

```
axlSetPlaneType(  
    t_subClassName  
    t_planeType  
)  
⇒ t/nil
```

Description

Changes the plane type of the conductor subclass.

Arguments

<i>t_subClassName</i>	Subclass name whose plane type is to be changed.
<i>t_planeType</i>	Plane type (Positive, Negative)

Value Returned

t	Plane type changed.
nil	Plane type is not changed.

Allegro PCB and Package User Guide: SKILL Reference
Parameter Management Functions

Selection and Find Functions

Overview

AXL edit functions such as move or delete operate on database object *dbids* contained in the *select set*. The select set is a list of one or more object *dbids* you have previously selected. You add *dbids* to the select set with the axlSelect functions described in this chapter. You use the select set as an argument in any edit function calls. This differs from interactive PCB Editor edit commands, where you first start a command, then select the objects to be edited. One advantage of a select set is that selected object *dbids* stay in the select set from function to function until you explicitly change it. You can perform multiple edits on the same set of objects without reselecting. Remember, however, that edit functions might cause *dbids* to go out of scope. This chapter also describes functions for managing the select set and controlling the selection filter.

AXL supports one active select set. The contents of the select set becomes invalid when you exit PCB Editor.

The interactive select functions find objects only on visible layers.

You can do the following with the selection functions:

- Set the Find Filter to control the types of objects selected
- Select objects at a single point, over an area, or by name
- Select parts of objects, such as individual pins of a package symbol
- Add or remove *dbids* from the select set

A select set can contain *dbids* of parts of a figure without containing the figure itself. For example, you can select one or more pins of a symbol or individual segments of a path figure. See [Chapter 2, “The PCB Editor Database User Model,”](#) for a list of PCB Editor figure types.

You can add figure *dbids* to the select set interactively with a mouse click on the figure (point selection) or by drawing a box that includes the figure (box selection). The **Find Filter** controls filtering for specific object types. This chapter describes AXL–SKILL functions to set the filter for any object type.

All coordinates are in user units unless otherwise noted.

Select Set Highlighting

Objects in the select set are displayed as highlighted when control passes from SKILL to you for interactive input (for selection) or when SKILL returns control to the PCB Editor shell. You can select and modify objects using AXL–SKILL functions. To keep these objects from displaying as highlighted, remove them from the select set before returning SKILL control to you for interactive input or to the PCB Editor shell.

Select Modes

AXL provides two select modes:

single	Selects one or a group of objects that match the selection criteria. When you select an object in this mode, AXL deselects any objects previously in the select set.
cumulated	Adds to or subtracts from the select set one or a group of objects that match the selection criteria. Add cumulated mode never adds an object already in the set, so you do not have duplicate entries if you mistakenly select the same object twice. Subsequent selects of the same object are ignored.

Finding Objects by Name

The `axlSingleSelectName`, `axlAddSelectName`, and `axlSubSelectName` functions find objects by using their names. You can search for the following PCB Editor object types by name:

- NET by netname
- COMPONENT by refdes
- SYMBOL by refdes
- FUNCTION by function designator
- DEVTYPE by device type
- SYMTYPE by symbol type (example "DIP14")

- PROPERTY by property name (example "MAX_OVERSHOOT")
- PIN by location (by giving <refdes><x_position>,<y_position>) or by <refdes>. <pin number> if assigned (example "REFDES" "U17.23")
- VIA by location (by giving <x_position>,<y_position>)

See the descriptions of the SelectName functions for how to set up the arguments.

Point Selection

These functions select objects at a single geometric point. They prompt you for the point if that argument is missing from the function call.

```
axlSingleSelectPoint  
axlAddSelectPoint  
axlSubSelectPoint
```

Area Selection

These functions select objects over an area. They prompt you for the area (*Bbox*) if that argument is missing from the function call.

```
axlSingleSelectBox  
axlAddSelectBox  
axlSubSelectBox
```

Miscellaneous Select Functions

These functions select by other means as shown in each function description later in this chapter:

```
axlAddSelectAll  
axlSubSelectAll  
axlSingleSelectName  
axlAddSelectName  
axlSubSelectName
```

`axlSingleSelectObject`

`axlAddSelectObject`

`axlSubSelectObject`

axlSelect—The General Select Function

One function, `axlSelect`, combines the capabilities of the `axlAddSelect` functions. Use `axlSelect` as you write interactive commands to give the user the same select capabilities available in PCB Editor commands *move* or *delete*.

Select Set Management

These functions manage the select set:

`axlGetSelSet`

`axlGetSelSetCount`

`axlClearSelSet`

Find Filter Control

These functions control the selection filter:

`axlGetFindFilter`

`axlSetFindFilter`

Find Filter Options

You can restrict selection in a given window to a subset of all possible figure types by using the **Find Filter** or the `FindFilter` functions described in this chapter. `FindFilter` functions also control whether the **Find Filter** is immediately visible to you.

The permissible keywords for the `lt_options` list are shown. These keywords are a subset of the PCB Editor Find Filter. You can prefix any of these keywords with `NO` to reverse the effect. See the description of [axlSetFindFilter](#) on page 237 for details on how to use these keywords:

<i>Global</i>	ALL, ALLTYPES, EQUIVLOGIC
<i>Figures</i>	PINS, VIAS, CLINES, CLINESEGS, LINES, LINESEGS, DRCS, TEXT, SHAPES (includes rectangles), SHAPESEGS, VOIDS, VOIDSEGS, TEXT.
Note:	The <code>xxxSEG</code> keywords also select arc and circle segments
<i>Logic</i>	COMPONENTS, SYMBOLS, NETS

Except for `EQUIVLOGIC`, the *Find Filter* functions take the keywords in the order found. For example, the list (`ALL NOPIN`) results in all objects except pins being selectable.

`EQUIVLOGIC` is a global find state. It instructs *find* to select the physically equivalent parts, as specified in the filter of the found logic object. For example, if the user picks any physical part of a net, the selection returns any physical parts of the (logical) net selected by the *find filter*. Both the logical and the physical objects must be enabled. For example, to select all pins of a net, both `NETS` and `PINS` must be enabled and set with `?onButton`.

Selection and Find Functions

This section lists selection and find functions.

axlSingleSelectPoint

```
axlSingleSelectPoint(  
  [ l_point ]  
)  
⇒ t/nil
```

Description

Clears the select set, finds a figure at *l_point* according to the Find Filter, and puts the selected figure *dbid* in the select set. If *l_point* is *nil*, the function requests a single pick from the user.

axlSingleSelectPoint selects a *single* object and adds it to the select set, unless **EQUIVLOGIC** is on. In that case, it may select multiple objects, for example, if it finds a qualified figure (such as a pin, connect line, or via) that is part of a net.

axlSingleSelectPoint adds all the qualified figures that belong to the net to the select set. (See the [Find Filter Options](#) on page 211 .)

Arguments

l_point Point in database coordinates at which to look for figures.

Value Returned

t One or more *dbids* put in the select set.

nil No *dbids* put in the select set.

Example

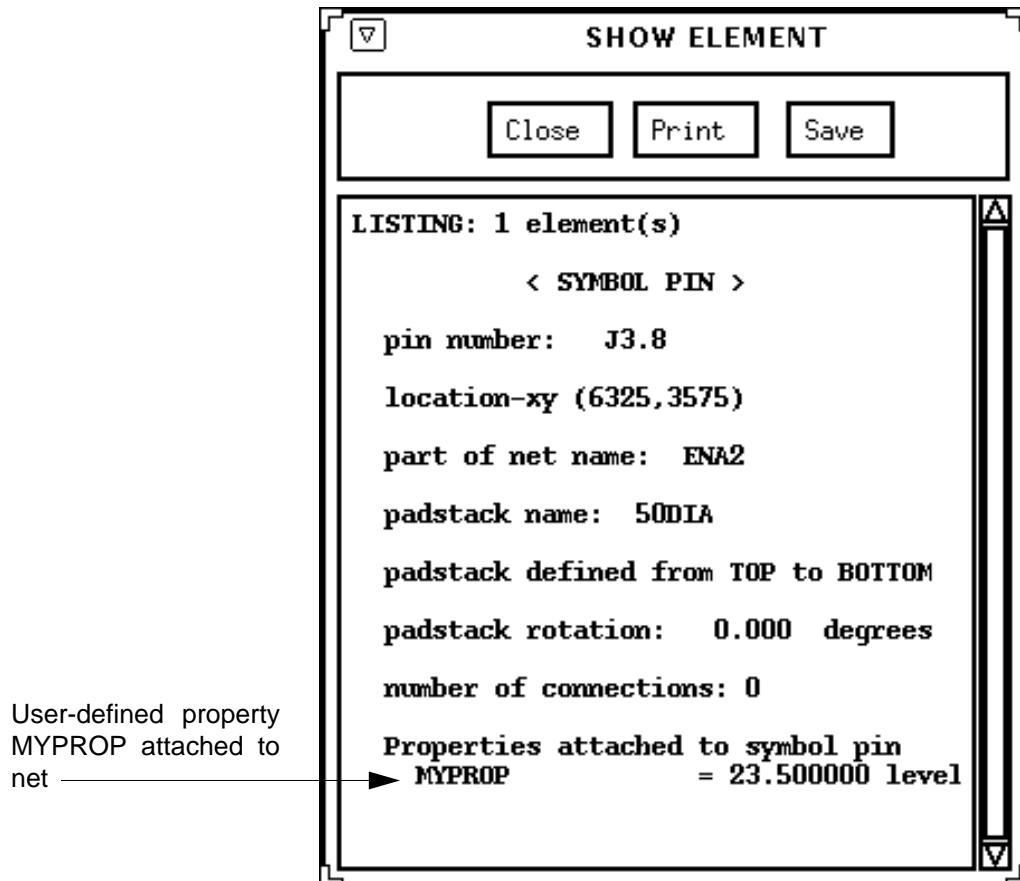
```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")
    axlSingleSelectPoint( 6325:3575 )
    axlDBAddProp(axlGetSelSet() list( "MYPROP" 23.5 ))
⇒ t
```

Adds a previously defined user property MYPROP to a pin at X6325 Y3575.

To check

1. From PCB Editor, Select *Display – Element*.
2. Select the pin to display its properties.

The **Show Element** window displays *MYPROP* with value *23.500000 level*



axlAddSelectPoint

```
axlAddSelectPoint(  
    [l_point]  
)  
⇒ t/nil
```

Description

Finds a figure at *l_point* according to the Find Filter and adds its *dbid* to the select set in cumulated mode. If *l_point* is *nil*, requests a single pick from the user.

Selects a *single* object and adds it to the select set, unless `EQUIVLOGIC` is on. In that case, selects multiple objects, for example, if it finds a qualified figure (such as a pin, connect line, or via) that is part of a net. Adds all the qualified figures that belong to the net to the select set. (See the [Find Filter Options](#) on page 211.)

Arguments

l_point Point in the layout at which to find figures.

Value Returned

t One or more *dbids* put in the select set.

nil No *dbids* put in the select set.

Example

See [axlSingleSelectPoint](#) on page 212 for an example. `axlSingleSelectPoint` has the same behavior except that it selects only one object.

axlSubSelectPoint

```
axlSubSelectPoint(  
    [l_point]  
)  
⇒ t/nil
```

Description

Finds a figure at *l_point* according to the Find Filter and deletes its *dbid* from the select set in cumulated mode. That is, it deletes that *dbid* while leaving any others currently in the select set. If *l_point* is nil, requests a single pick from the user.

Removes a *single* object from the select set, unless EQUIVLOGIC is on. In that case, finds multiple objects, for example, if it finds a qualified figure (such as a pin, connect line, or via) that is part of a net. Deletes all the qualified figures that belong to the net from the select set. (See the [Find Filter Options](#) on page 211 .)

Arguments

l_point Point in the layout at which to find figures to deselect.

Value Returned

t One or more *dbids* removed from the select set

nil No *dbids* removed from the select set.

Example

```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")  
axlSingleSelectBox( list(6200:3700 6500:3300))  
axlSubSelectPoint( 6325:3575 )  
axlDBAddProp(axlGetSelSet() list("MYPROP" 23.5))  
⇒ t
```

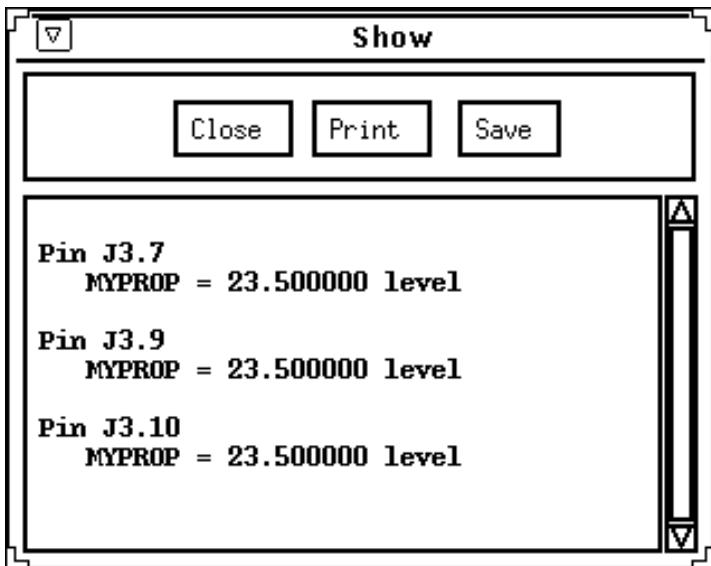
Adds a previously defined user property, MYPROP

1. Selects four pins in a box in order.
2. Before adding the property, subtracts the pin at 6325 : 3575 from the list, then adds as shown.

To check

1. Select *Edit – Properties*.
2. Select a window around all four pins.
3. Select *MYPROP* from the Available Properties list in the **Edit Property** window.

The command displays the pins that have properties in the **Show Properties** window. Only the three pins not subtracted from the select set have the property, *MYPROP*.



axlSingleSelectBox

```
axlSingleSelectBox(  
    [ l_bBox ]  
)  
⇒ t/nil
```

Description

Clears the select set, finds all figures inside the rectangle *l_bBox* according to the Find Filter, and adds the selected figure *dbids* in single mode to the select set.

Arguments

<i>l_bBox</i>	List containing one or two coordinates defining the bounding box to be used to select the figures. If <i>l_bBox</i> is nil, requests two picks from the user. If <i>l_bBox</i> has only one point, asks for a second point from the user.
---------------	---

Value Returned

t	One or more objects added to the select set.
---	--

nil	No objects added to the select set.
-----	-------------------------------------

Example

```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")  
axlSingleSelectBox( list(6200:3700 6500:3300))  
axlDBAddProp(axlGetSelSet() list( "MYPROP" 23.5))  
⇒ t
```

Adds a previously defined user property MYPROP to four pins by selecting a box around them with corners (6200:3700 6500:3300).

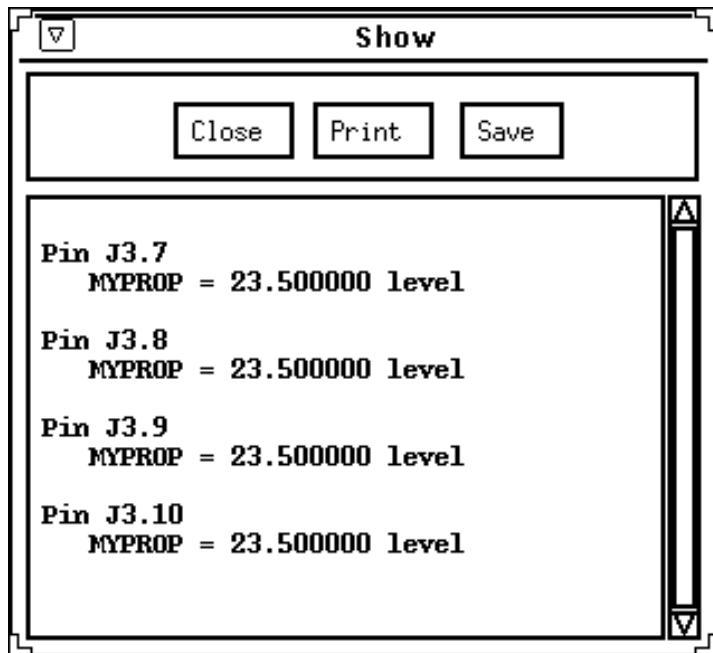
To check

1. Select *Edit – Properties*.
2. Select the four pins.
3. Select *MYPROP* from the Available Properties list in the **Edit Property** window.

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

The command displays the four pins in the **Show Properties** window as having **MYPROP** with value **23.500000 level**



axlAddSelectBox

```
axlAddSelectBox(  
    [ l_bBox ]  
)  
⇒ t/nil
```

Description

Finds one or more figures inside the rectangle *l_bBox* according to the current Find Filter, and adds the selected figure *dbids* in cumulated mode for the select set.

Arguments

<i>l_bBox</i>	List containing one or two coordinates defining the bounding box to be used to the select figures. If <i>l_bBox</i> is <i>nil</i> , requests two picks from the user. If <i>l_bBox</i> has only one point, asks for a second point from the user.
---------------	---

Value Returned

t	One or more objects added to the select set.
---	--

nil	No objects added to the select set.
-----	-------------------------------------

Example

See the example for [axlSingleSelectBox](#) on page 217. That function behaves exactly as `axlAddSelectBox`, except that `axlSingleSelectBox` does not clear the select set.

axlSubSelectBox

```
axlSubSelectBox(  
    [ l_bBox ]  
)  
⇒ t/nil
```

Description

Finds one or more figures inside the rectangle *l_bBox* according to the Find Filter, and deletes their *dbids* from the select set in *cumulated* mode. Deletes those *dbids* while leaving any others currently in the select set.

Arguments

l_bBox List containing one or two coordinates defining the bounding box to be used to the select figures. If *l_bBox* is *nil*, requests two picks from the user. If *l_bBox* has only one point, asks for a second point from the user.

Value Returned

t One or more objects deleted from select set.

nil No objects deleted from select set.

Example

See [axlSubSelectPoint](#) on page 215 for an example of subtracting from the select set, and [axlAddSelectPoint](#) on page 214 for an example of using a box for selection.

axlAddSelectAll

```
axlAddSelectAll()  
)  
⇒ t/nil
```

Description

Finds all the figures in the database that pass the current Find Filter and adds their *dbids* to the select set.

Arguments

None.

Value Returned

t	One or more object <i>dbids</i> added to the select set.
nil	No object <i>dbids</i> added to the select set.

Example

```
axlSetFindFilter(?enabled list( "noall" "vias")  
?onButtons list( "noall" "vias"))  
axlAddSelectAll()  
axlDeleteObject(axlGetSelSet())  
⇒ t
```

Selects all vias in a layout and deletes them.

axlSubSelectAll

```
axlSubSelectAll()  
)  
⇒ t/nil
```

Description

Finds all figures in the database that pass the current Find Filter and deletes their *dbids* from the select set. Use `axlSubSelectAll` to subtract all of a given type of figure from a larger set of selected objects.

Note: Use `axlClearSelSet` to deselect all figures in the current select set, regardless of the current Find Filter.

Arguments

None.

Value Returned

t	One or more <i>dbids</i> deleted from select set.
nil	No <i>dbids</i> deleted from select set.

Example

```
axlClearSelSet()  
axlSetFindFilter(?enabled  
    list( "noall" "equivlogic" "nets" "clines" "vias" )  
    ?onButtons list( "all" ))  
axlAddSelectPoint()  
axlSetFindFilter(?enabled list( "noall" "vias" )  
    ?onButtons list( "all" ))  
axlSubSelectAll()  
axlDeleteObject(axlGetSelSet())  
⇒ t
```

Interactively selects all connect lines (clines) and vias on a net, subtracts the via *dbids* from the select set, and deletes the remaining *dbids* in the select set.

The prompt *Enter selection point*, is displayed. Only the connect lines on the net you select are deleted—not the vias of the net.

axlSingleSelectName

```
axlSingleSelectName(  
    t_nameType  
    l_names  
)  
⇒ t/nil
```

Description

Finds figures by their names. Clears the current contents of the select set, and adds named figure *dbids* to the select set in single mode using the arguments as described below. The function selects any figures completely, regardless of the selection mode. If the function selects a figure already partially selected, the figure becomes fully selected.

Note: The *on* buttons are used for selecting *Properties* by name only.

Arguments

<i>t_nameType</i>	String denoting the name type to be selected (see Finding Objects by Name on page 208).
<i>l_names</i>	One of three possibilities (See examples): -Name -List of names -List of name/value pairs

Value Returned

<i>t</i>	One or more objects added to the select set.
<i>nil</i>	No objects added to the select set.

Example 1

```
axlClearSelSet()
axlSetFindFilter(
    ?enabled list( "noall" "nets")
    ?onButtons list( "all"))
axlSingleSelectName ("NET" (list "GND" "VCC"))
⇒ (dbid:234436 dbid:98723)
    axlSingleSelectName ("PROPERTY" (list (list "BUS_NAME" "MEM") "FIXED"))
```

Selects the nets GND and VCC by their names.

Example 2

```
axlClearSelSet()
axlSetFindFilter(
    ?enabled list( "noall" "nets")
    ?onButtons list( "all"))
axlSingleSelectName ("PROPERTY"
    list(
        list( "ELECTRICAL_CONSTRAINT_SET" "ECL_DEFAULT"
            "ROUTE_PRIORITY"))
⇒ (dbid:234436 dbid:98723 dbid:234437 dbid:98727
    dbid:234438 dbid:98725 dbid:234439 dbid:98726)
```

Selects a set of nets—one set by the property name ELECTRICAL_CONSTRAINT_SET with value ECL_DEFAULT, and another set that has the name ROUTE_PRIORITY.

axlAddSelectName

```
axlAddSelectName(  
    t_nameType  
    l_names  
)  
⇒ t/nil
```

Description

Adds the named figure *dbids* to the select set in cumulated mode according to the arguments described below. Adds the found figures to the select set if not already there. Selects figures completely regardless of the selection mode. If the function selects a figure already partially selected, the figure becomes fully selected.

Arguments

<i>t_nameType</i>	String denoting the name type to be selected (see Finding Objects by Name on page 208).
<i>l_names</i>	One of three possibilities (See examples): A name A list of names A list of name/value pairs

Value Returned

t	One or more objects added to the select set.
nil	No objects added to the select set.

Example

See examples for [axlSingleSelectName](#) on page 223. The only difference is that `axlSingleSelectName` clears the select set, while `axlAddSelectName` adds the selected *dbids* into the select set without removing any already in it. The arguments for `axlAddSelectName` are the same as for `axlSingleSelectName`.

axlSubSelectName

```
axlSubSelectName(  
    t_nameType  
    l_names  
)  
⇒ t/nil
```

Description

Removes *dbids* of the named figure from the select set using the arguments described. Removes figures completely regardless of the selection mode. If the function finds a figure already partially selected, it removes all of its *dbids* from the select set.

Arguments

<i>t_nameType</i>	String denoting name type to be selected (see Finding Objects by Name on page 208).
<i>l_names</i>	One of three possibilities (See examples): -Name -List of names -List of name/value pairs

Value Returned

t	One or more objects deleted from select set.
nil	No objects deleted from select set.

Example

See examples for [axlSingleSelectName](#) on page 223. The only difference is that `axlSingleSelectName` clears the select set and then puts the *dbids* it finds into the select set, while `axlSubSelectName` removes the *dbids* of the elements it selects from the select set. The arguments are the same as `axlSingleSelectName`.

axlSingleSelectObject

```
axlSingleSelectObject(  
    lo_dbid  
)  
⇒ t/nil
```

Description

Clears contents of the select set and adds the *dbids* in *lo_dbid* to the select set in single mode. *lo_dbid* is either a single *dbid* or a list of *dbids*. Selects figures completely regardless of the selection mode. If the *dbid* of any part of a figure is in *lo_dbid*, the function adds the entire figure.

Arguments

lo_dbid *dbid*, or list of *dbids* to be added to the select set.

Value Returned

t One or more objects added to the select set.

nil No objects added to the select set.

Example

```
axlClearSelSet()  
axlSetFindFilter(  
    ?enabled list( "all" "equivlogic")  
    ?onButtons list( "all"))  
mysym = axlDBCreateSymbol(  
    list( "dip14" "package"), 5600:4600)  
axlSingleSelectObject(car(mysym))  
⇒ t
```

Creates a symbol and add its *dbid* to the select set.

axlAddSelectObject

```
axlAddSelectObject(  
    lo_dbid  
)  
⇒ t/nil
```

Description

Adds the *dbids* in *lo_dbid* to the select set in cumulated mode, that is, without removing already selected objects. *lo_dbid* is either a single *dbid* or a list of *dbids*. Adds *dbids* in the select set only if they are not already there. Selects figures completely regardless of the selection mode. If the *dbid* of any part of a figure is in *lo_dbid*, adds the entire figure.

Arguments

lo_dbid *dbid*, or list of *dbids* to be added to the select set.

Value Returned

t One or more objects added to the select set.

nil No objects added to the select set.

Example

```
axlSetFindFilter(  
    ?enabled list( "all")  
    ?onButtons list( "all"))  
mysym = axlDBCreateSymbol(  
    list("dip14" "package") 2600:2600 ) axlAddSelectObject(car(mysym))  
⇒ t
```

Creates a symbol instance and adds its *dbid* to the select set.

axlSubSelectObject

```
axlSubSelectObject(  
    lo_dbid  
)  
⇒ t/nil
```

Description

Removes the *dbids* in *lo_dbid* from the select set in cumulated mode. *lo_dbid* is either a single *dbid* or a list of *dbids*. Removes figures completely regardless of the selection mode.

Arguments

lo_dbid *dbid*, or list of *dbids* to be removed from select set.

Value Returned

t One or more objects deleted from select set.

nil No objects deleted from select set.

Example

```
axlClearSelSet()  
axlSetFindFilter(?enabled list("noall" "vias")  
    ?onButtons list( "vias"))  
myvia = axlDBCreateVia( "pad1", 3025:3450)  
axlAddSelectBox( list( 3000:3100 3200:3600))  
axlSubSelectObject( car( myvia))  
⇒ t
```

Creates a via, then selects all the vias in a surrounding region for deletion, but saves the one just created by subtracting its *dbid* from the selection set.

The resulting select set contains the *dbids* of all vias in the box except *myvia*, the one just created.

axlSelect

```
axlSelect(  
    ?firstEventCallback      s_callback  
    ?groupMode               t/nil  
    ?prompt                  t_prompt  
)  
⇒ t/nil
```

Description

General tool for AXL programs to solicit interactive object selections from the user. axlSelect automatically sets up the pop-up to provide any of the possible PCB Editor selection methods:

- Single point select
- Window select
- Group select

You can set up the pop-up to display other options such as *Done* and *Cancel*, but the function also displays the find options. See the example.

Before axlSelect returns, it puts in the select set a list of the *dbids* of the objects the user selected.

Use axlSelect when you create interactive commands that allow the user to select objects in the same way as existing PCB Editor interactive commands such as *move* or *delete*. axlSelect allows the user to select objects using the standard methods of mouse pick, window, and group. It returns when the user has selected one or more objects or picks *Done* or *Cancel*, depending on the pop-up selections you set up. The default mode for axlSelect is selecting a single object by point—equivalent to axlSingleSelectPoint.

axlSelect removes any previously selected *dbids* in the selection set when the user first selects one or more objects.

You must set up the find filter to meet your requirements before calling axlSelect.

Arguments

<i>firstEventCallback</i>	Procedure to be called once the first user event occurs. The callback takes no arguments.
---------------------------	---

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

<i>groupMode</i>	Default is for <code>axlSelect</code> to return when the user makes a selection. If <code>groupMode</code> is ' <code>t</code> ', <code>axlSelect</code> won't return until you do an <code>axlCancelEnterFun</code> or <code>axlFinishEnterFun</code> . You perform all of your activity in popup callbacks instead of when <code>axlSelect</code> returns. In <code>groupMode</code> , <code>axlSelect</code> does not clear existing selections. To clear existing selections after the first event, clear them in your <code>firstEventCallback</code> .
<i>prompt</i>	Prompt to the user. The default prompt is: "Enter selection point"

Value Returned

<code>t</code>	One or more object <code>dbids</code> put into the select set during the call. The select set is a list of the <code>dbids</code> of the objects the user selected.
<code>nil</code>	No object <code>dbids</code> put into the select set.

Example

```
(defun showElement ()
  mypopup = axlUIPopupDefine( nil
    (list (list "Done" 'axlFinishEnterFun)
          (list "Cancel" 'axlCancelEnterFun)))
  axlUIPopupSet( mypopup)
  axlSetFindFilter( ?enabled list( "NOALL" "ALLTYPES" "NAMEFORM")
    ?onButtons "ALLTYPES")
  while( axlSelect()
    axlShowObject( axlGetSelSet())))
```

Function loops, performing the `Show Element` command on each object selected by the user.

Although you explicitly set *Done* and *Cancel* for this command, AXL also adds *Group*, *Window Select*, and *FindFilter* to the pop-up.

Done
Cancel
Group
Window Select
Find Filter

axlGetSelSet

```
axlGetSelSet()  
)  
⇒ lo_dbid/nil
```

Description

Gets the list of object *dbids* in the select set.

Arguments

None.

Value Returned

<i>lo_dbid</i>	List of figure <i>dbids</i> .
<i>nil</i>	Select set is empty.

Example

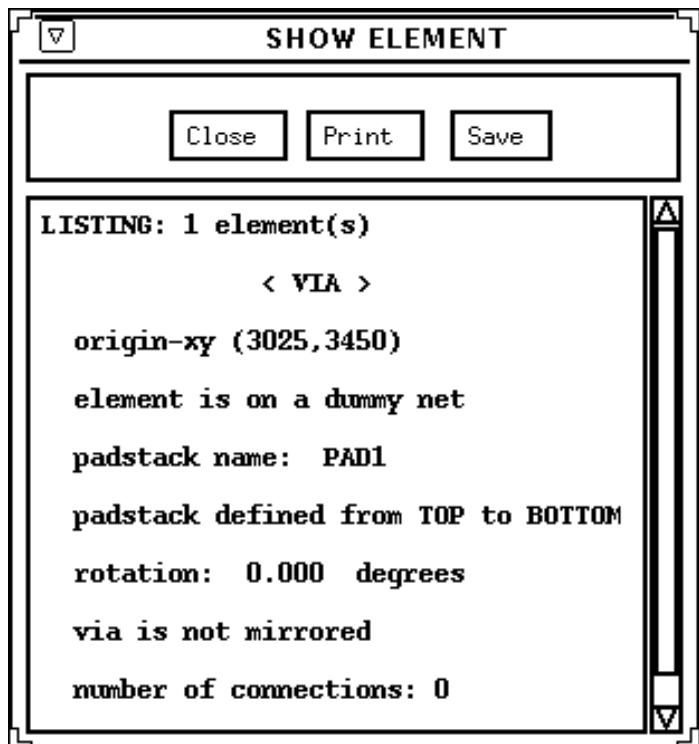
```
axlClearSelSet()  
axlSetFindFilter(?enabled list("noall" "vias")  
    ?onButtons list("vias"))  
axlAddSelectBox(list(3000:3100 3200:3600))  
axlShowObject(axlGetSelSet())  
⇒ t
```

Selects all vias in a box area and shows the contents of the select set.

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

The **Show Element** window is displayed, with the via selected.



axlGetSelSetCount

```
axlGetSelSetCount()  
)  
⇒ x_selCount
```

Description

Returns the number of figure *dbids* in the select set.

Arguments

None.

Value Returned

x_selCount	Number of objects selected. Returns 0 if nothing is selected.
------------	---

Example

```
axlClearSelSet()  
axlSetFindFilter(?enabled list("noall"))  
axlSetFindFilter(?enabled list("pins")  
    ?onButtons list("pins"))  
axlAddSelectBox()  
axlGetSelSetCount()  
⇒ 14
```

Sets the Find Filter to find pins, selects a box around a 14 pin dip and prints the number of *dbids* in the select set. It is 14, as expected.

axlClearSelSet

```
axlClearSelSet()  
)  
⇒ t/nil
```

Description

Removes all *dbids* from select set.

Note: Use `axlDeselectAll` to deselect all with respect to Find Filter setting.

Arguments

None.

Value Returned

t	One or more <i>dbids</i> removed in order to empty the select set.
nil	Select set already empty.

Example

```
axlClearSelSet()  
axlSetFindFilter(?enabled list("noall"))  
axlSetFindFilter(?enabled list("pins"))  
    ?onButtons list("pins"))  
axlAddSelectBox()  
axlGetSelSetCount()  
⇒ 14
```

Ensures the select set does not have any leftover *dbids* in it, as in the example for `axlGetSelSetCount`.

axlGetFindFilter

```
axlGetFindFilter(  
    [onButtonF]  
  
)  
⇒ lt_filters/nil
```

Description

Gets the Find Filter settings and returns them as keyword strings in the list *lt_filters* according to the value of *onButtonF*, as described.

Arguments

onButtonF If *onButtonF* is t, returns the enabled list. If *onButtonF* is nil, returns the *onButton* list.

Value Returned

lt_filters List of Find Filter settings (see the [Find Filter Options](#) on page 211).

nil

Example

```
(axlSetFindFilter ?enabled (list "vias" "pins" "nets" "clinesegs" "nameform")  
?onButtons (list "vias" "pins" "clinesegs"))  
(axlGetFindFilter)
```

Returns the following:

```
( "NAMEFORM" "NETS" "CLINESEGS" "VIAS" "PINS" )
```

axlSetFindFilter

```
axlSetFindFilter(  
    ?enabled      lt_enabled  
    ?onButtons    lt_filterOn  
  
)  
t/nil
```

Description

Sets up both the object types to be displayed in the Find Filter, and which types among those are set to *on* in the **Find Filter**.

The first argument, *lt_enabled*, is a list of the object types to be displayed in the Find Filter and of the select options described. The second argument, *lt_onButtons*, lists the object types whose buttons are to be *on* (and therefore selectable) when the filter displays. The table lists the keywords that can be included in the enabled and *onButtons* lists for setting up the Find Filter. The diagrams show the keywords and the buttons they cause *axlSetFindFilter* to display.

Each change is additive and processed in the order that they appear in the list. For example, you type the following to enable all object types except for pins.

```
' ( "ALLTYPES"  "NOPINS" )
```

Each of the following keywords may be preceded with a "NO" to disable the particular option or object type. For example, "NOPINS". The initial default is "NOALL".

axlSetFindFilter Keywords

Keyword	Description
"PINS"	Enable pins
"VIAS"	Enable vias
"CLINES"	Enable clines
"CLINESEGS"	Enable cline (arc or line) segs
"LINES"	Enable lines
"LINESEGS"	Enable line (arc or line) segs
"DRCS"	Enable drc errors
"TEXT"	Enable text

axlSetFindFilter Keywords, continued

Keyword	Description
"SHAPES"	Enable shapes, rects and frects
"SHAPESEGS"	Enable shape segments
"BOUNDARY_SHAPES"	Enable promotion to boundary shape if auto-shape is selected (see dynamic shape discussion)
"VOIDS"	Enable shape voids
"VOIDSEGS"	Enable shape void segments
"SYMBOLS"	Enable symbol instances
"FIGURES"	Enable figures
"COMPONENTS"	Enable component instances
"FUNCTIONS"	Enable function instances
"NETS"	Enable nets
"AUTOFORM"	Option - OBSOLETE
"EQUIVLOGIC"	Option - For logic object types (nets and components), causes the physical equivalent to be selected. Physical objects must be enabled in both "enabled" and "onButton" lists.
"INVISIBLE"	Option - Allows selection of objects that are not visible due to color <i>class</i> / <i>subclass</i> form setting.
"NAMEFORM"	Option - Enables the <i>Find by Name/property</i> fields in the Find Filter form.
"ALLTYPES"	Enables all object types ("PINS" . . . "NETS").
"ALL"	Enables all object types and options.
"DYNTHEMALS"	Enable selection of themal reliefs generated by dynamic shapes. Only applicable to ?enabled. By default, you should not select these if you plan on modifying the objects since the dynamic shape will just re-generate them. You should only access them for read-only purposes.

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

`axlSetFindFilter` Keywords, continued

Keyword	Description			
"GROUPS"	"GROUPS" and "GROUPMEMBERS" operate together to produce four possible selection states:			
"GROUPMEMBERS"				
"GROUPS" and "GROUPMEMBERS" States				
Keyword	1	2	3	4
GROUPS	OFF	ON	OFF	ON
GROUPMEMBERS	OFF	OFF	ON	ON
State 1	Legacy. This supports code that predates the group implementation.			
State 2	Group only. By only setting the group bitfield, any selected group is returned to the application as a group.			
State 3	Members only. By only setting the group_members bitfield, group members are returned to the application when a group is selected.			
State 4	Hierarchical. By setting both the group and group_members bitfields, a group is returned for any hierarchical group that is selected (such as a Module instance), and group members are returned for all other selected group types.			

`axlSetFindFilter` processes the keywords in each argument list in order and only makes the changes occurring in the list. Changes are incremental for each call to the function. To

remove a selection, attach the string "NO" to the front of the keyword. For example, the list ("ALLTYPES" "NOPINS") enables all object types except pins. The initial default setting of the Find Filter is "NOALL", or, nothing enabled. Use "NOALL", as shown, to clear the Find Filter before enabling particular types.

Dynamic Shapes

When writing an application and it needs to handle shapes, you need to decide how you want to handle dynamic shapes. If your SKILL program does not access shapes, read no further.

A shape on ETCH may be either static or dynamic. A static shape is similar to what existed in PCB Editor prior to PSD15.0. You add a shape to an etch layer and manually void objects that impact the shape. A dynamic shape is placed on the BOUNDARY CLASS and generates zero or more shapes on the ETCH layer based upon voiding.

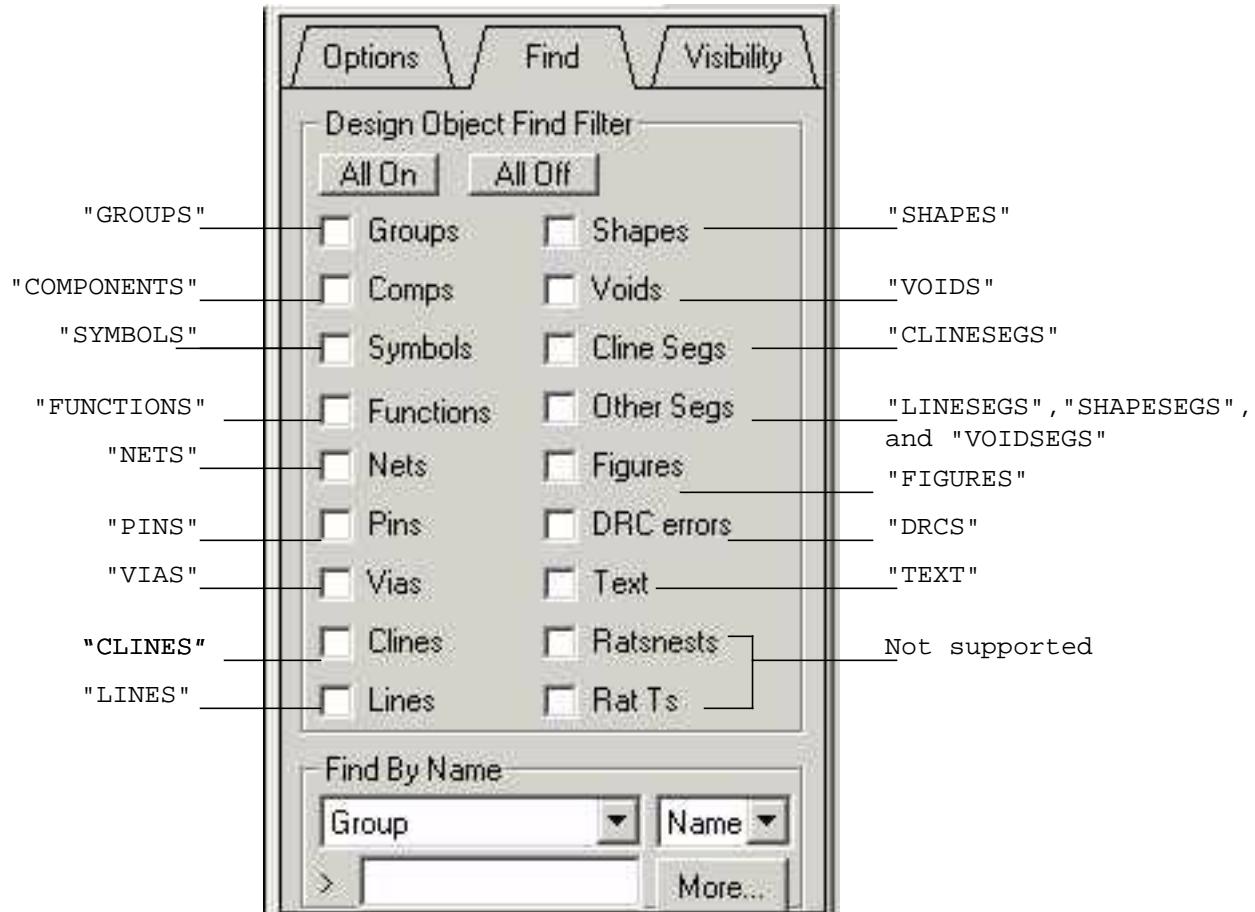
If you want to modify a dynamic shape, then you should set BOUNDARY_SHAPES. This allows the user to select the generated shape, but selection will return its dynamic shape (e.g. a *move shape*). If you want to access information on the shape, then do not set the BOUNDARY_SHAPES option (e.g. *show element*).

Note: If you pass "ALL" or "ALLTYPES" to *setOptions*, then BOUNDARY_SHAPES will be enabled and user's selecting a ETCH layer generated shape will result in the selection returning its dynamic shape on the "BOUNDARY CLASS". If you wish to select the generated shape, but want to use the *all* option, then use the following:

```
"(ALLTYPES" "NOBOUNDARY_SHAPES")
```

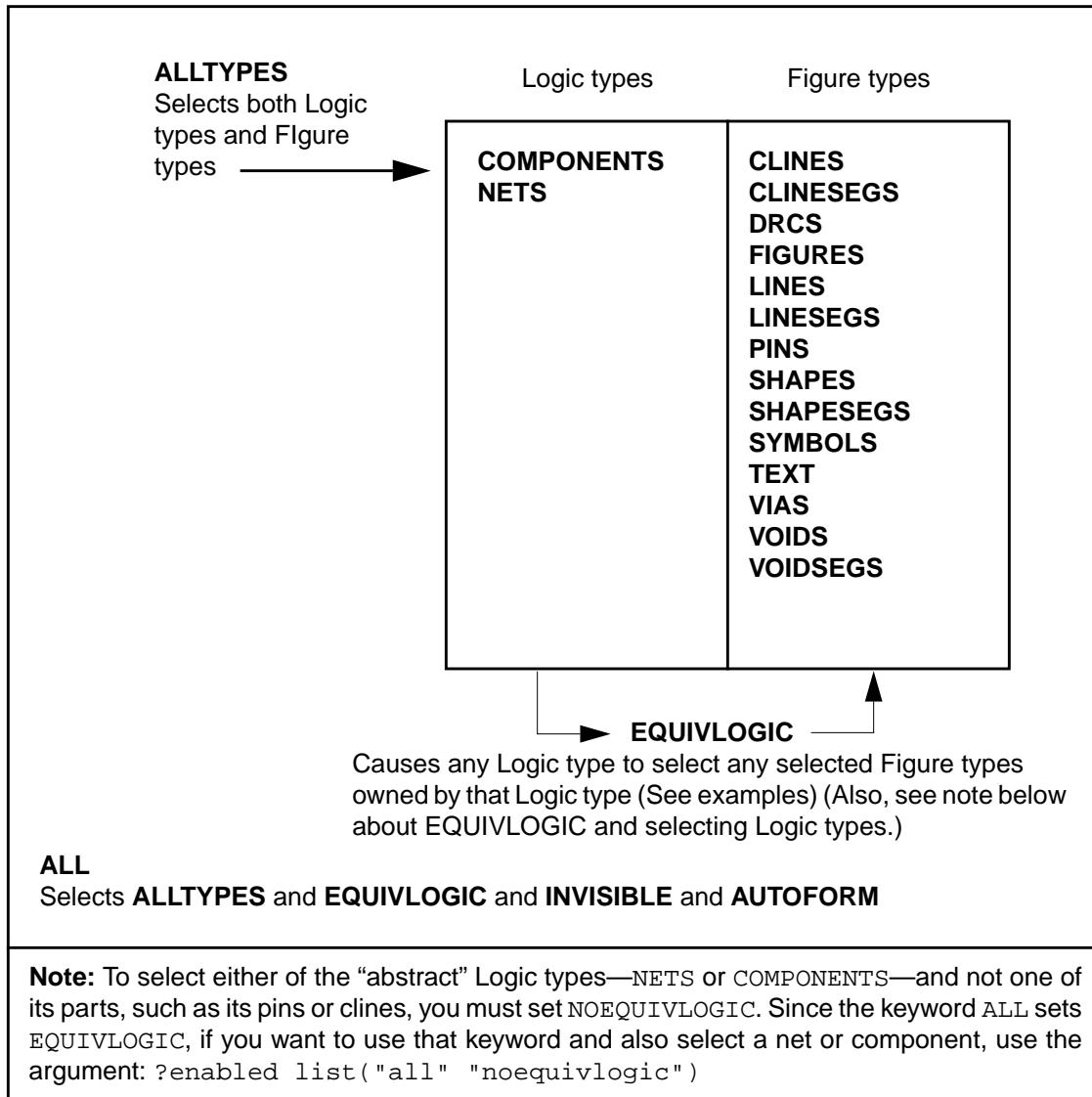
You need only pass BOUNDARY_SHAPES to the enabled list. It will be ignored if passed to the onButtons list.

Figure 5-1 Find Filter and Related Keywords



The differences in effects of different combinations of keywords can be subtle. [Figure 5-2](#) on page 242 shows the relationship between the keywords.

Figure 5-2 Relationship Among axlSetFindFilter Keywords (See examples also)



Arguments

<code>lt_enabled</code>	List of keyword strings that describe object types that are to be selectable. Enabled object types will appear in the Find Filter form. Object types need the <code>onButton</code> set as well to fully enable selection. List may also include selection options. Also supports a single keyword string instead of a list of strings.
-------------------------	---

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

`lt_onButtons`

List of keyword strings that describe object types that are to be enabled for selection. Enabled types will appear with the `onButton` depressed in the Find Filter form. `onButton` settings provide the default for controls which can be modified by the user when the Find Filter form is opened. An object type must be `on` in both the enabled and the `onButton` lists to be fully enabled for selection. Options are ignored when provided in the `onButton` list.

Also supports a single keyword string instead of a list of strings.

Note: `axlSetFindFilter` does not display or select any types that are not enabled. That means that `?onButtons` keywords only effect enabled types. For example, to have all enabled buttons be `on`, use `?onButtons list("alltypes")`. In general, you need only set specific buttons `on` if you want those on and others off.

Value Returned

`t` One or more Find Filter changes were made.

`nil` No valid keywords were provided.

Application Programming Note

When you use `axlSetFindFilter` to implement an interactive command, make your AXL-SKILL program restore the user's FindFilter settings from the previous time he or she used the same command. Find Filter settings are incremental. Clear any previous settings as you start, then set the ones you want. Call `axlSetFindFilter` with "noall" as the first member of the list for both the `?enabled` and `?onButtons` arguments the first time you call it.

To maintain the user's Find Filter settings between invocations of your command

1. The first time the user invokes your program (when it loads), preset global variables to the list of `?enabled` and `?onButtons` settings you want as the initial default, as follows:

```
myglobal_enabled = list("noall" "xxx" "yyy" ... "zzz")
myglobal_onButtons = list("xxx" ... "zzz")
```

2. Each time the user invokes your command, call `axlSetFindFilter` with the current global values of `?enabled` and `?onButtons`.

```
axlGetFindfilter( ?enabled myglobal_enabled
                  ?onButtons myglobal_onButtons)
```

Since users can set or clear any of the buttons on the Find Filter, you need to save the button settings as you exit the command.

3. As you end the command, save the user's Find Filter `onButton` settings as shown:

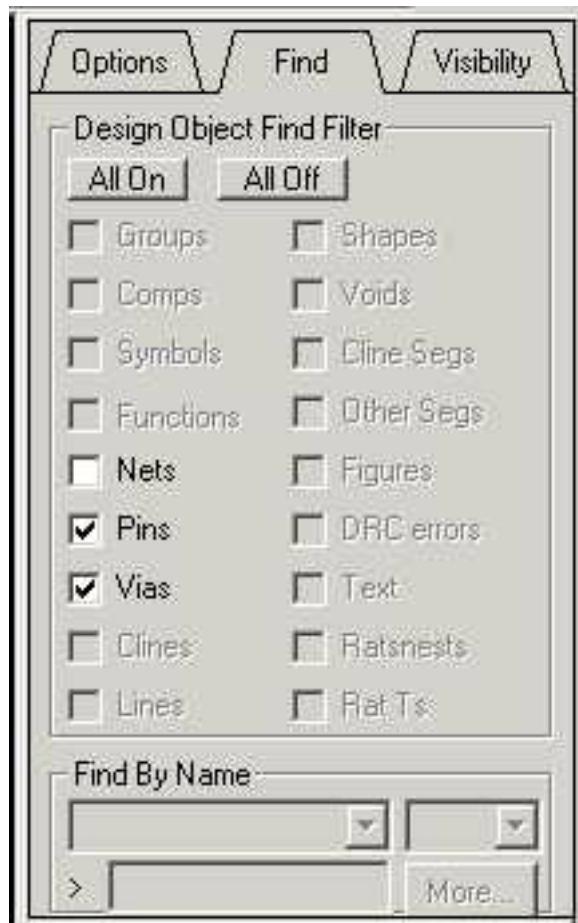
```
myglobal_onButtons = cons( "noall" axlGetFindFilter(t))
```

Note: The `cons` - "noall" ensures that when you call `axlSetFindFilter` again you clear any settings left over from any previous command, and set only the buttons set at the time you call `axlGetFindFilter`.

Example 1

```
(axlSetFindFilter ?enabled (list "vias" "pins" "nets")
?onButtons (list "vias" "pins"))
```

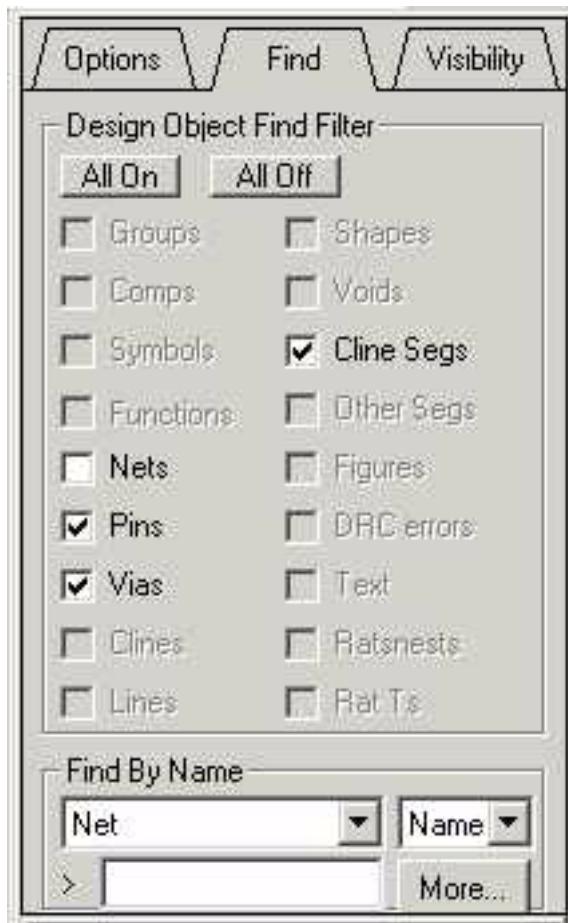
Displays the Find Filter with a list of Nets, Pins, and Vias. The Pins and Vias boxes are turned on as shown:



Example 2

```
(axlSetFindFilter ?enabled (list "noall" "vias" "pins"  
"nets" "clinesegs" "nameform")  
?onButtons (list "vias" "pins" "clinesegs"))
```

Displays the Find Filter with Pins, Vias, and Clinesegs turned on.



Example 3

```
axlSetFindFilter(  
?enabled list("noall" "equivlogic" "nets" "pins" "vias")  
?onButtons list("all"))
```

Sets to find all the pins and vias on a net when the user selects any part of the net.

axlAutoOpenFindFilter

```
axlAutoOpenFindFilter()  
)  
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

t	Returns t always.
---	-------------------

axlOpenFindFilter

```
axlOpenFindFilter(  
)  
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

t	Returns t always.
---	-------------------

axlCloseFindFilter

```
axlCloseFindFilter()  
)  
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

t	Returns t always.
---	-------------------

axlFindFilterIsOpen

```
axlFindFilterIsOpen()  
)  
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

t	Returns t always.
---	-------------------

axlSelectByName

```
axlSelectByName (
    t_objectType
    t_name / lt_name
    [g_wildcard]
)
⇒ lo_dbid/nil
```

Description

Selects database objects by name.

Interface allows more than one name to be passed, but only one object type per call. For certain object types, a single name may return multiple objects. The supported object types and related items follow:

Object Type	Item the function finds
"NET"	net name
"COMPONENT"	component name
"FUNCTION"	function name
"DEVTYPE"	device type name
"SYMTYPE"	symbol type name
"PIN"	refdes.pinname
"REFDES"	find symbol by refdes name
"COMPREFDES"	find component by refdes name
"GROUP"	find group by name
"BUS"	find bus by name
"DIFF_PAIR"	find differential pair by name
"XNET"	find xnet by name
"MATCH_GROUP"	find matchgroup by name
"MODULE"	find module by name

You can use wildcards with this function:

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

- “*” matches any sequence of zero or more characters.
- “?” matches any single character.
- “\” is used to send a special character, for example, \x.

Note: This saves and restores the current find filter settings, but resets the selection set.

Arguments

t_objectType Type of database name.

t_name Object to find.

lt_name List of names to find.

Value Returned

t List of objects found.

nil No matching name or illegal *objectType* name.

Example 1

```
Skill > p = axlSelectByName( "NET"  ' ( "GND"  "NET1" ) )
(dbid:28622576 dbid:28639844)
```

Finds two nets.

Example 2

```
Skill > p = axlSelectByName( "NET"  ' ( "GND"  "FOO" ) )
(dbid:28622576)
```

Finds two nets, but board only has GND.

Example 3

```
Skill > p = axlSelectByName( "COMPONENT"  "C1" )
(dbid:28590388)
Skill > car(p)->objType
"component"
```

Finds Component C1.

Example 4

```
Skill > p = axlSelectByName( "FUNCTION"  "TF-326" )
(dbid:28661572)
Skill > car(p)->objType
"function"
```

Finds function TF-326.

Example 5

```
Skill > p = axlSelectByName( "DEVTYPE" "CAP1" )
(dbid:28591032 dbid:28590700 dbid:28590388)
Skill > car(p)->objType
"component"
```

Finds devices of type CAP1.

Example 6

```
Skill > p = axlSelectByName( "SYMTYPE" "dip14_3" )
(dbid:28688416 dbid:28686192)
Skill > car(p)->objType
"symbol"
```

Finds symbols of type DIP14_3.

Example 7

```
Skill > p = axlSelectByName( "PIN" "U1.1" )
(dbid:28630692)
Skill > car(p)->objType
"pin"
```

Finds pin U2.1.

Example 8

```
Skill > p = axlSelectByName( "REFDES" "U3" )
(dbid:28688416)
Skill > car(p)->objType
"symbol"
```

Finds symbol by refdes U3.

Example 9

```
Skill > p = axlSelectByName( "COMPREFDES" "U3" )
(dbid:28621208)
Skill > car(p)->objType
"component"
```

Finds component by refdes U3.

Example 10

```
Skill > p = axlSelectByName( "GROUP" "BAR" )
(dbid:28593776)
Skill > car(p)->objType
```

Finds a group BAR by name.

Example 11

```
Skill > p = axlSelectByName( "PIN" "U1.*" )
(dbid:28630856 dbid:28630784 dbid:28630692 dbid:28630572 dbid:28630400
dbid:28630228 dbid:28630076 dbid:28630004 dbid:28629912 dbid:28629800
dbid:28629728 dbid:28629656 dbid:28629484 dbid:28629372 dbid:28629280
dbid:28629208
)
```

Finds all pins on refdes U1.

Example 12

```
Skill > p = axlSelectByName( "PIN" "U1.?" )
(dbid:28630856 dbid:28630692 dbid:28630572 dbid:28630400 dbid:28630228
dbid:28630076 dbid:28630004 dbid:28629912 dbid:28629800
)
```

Finds pins with single digit number on U1.

axlSelectByProperty

```
axlSelectByProperty(  
    t_objectType  
    t_property  
    [t_value]  
    [gRegularExpression]  
)  
⇒ lo_dbid/nil
```

Description

Selects the *dbid* set of a particular PCB Editor database object with the indicated property.

Property can be a name or a name/value pair. Value may contain a regular expression (* or ?), since certain select by name functions support wildcards. You can test for the presence of wildcards before you call this function.

Regular expressions used by PCB Editor differ from the Skill regular expressions. PCB Editor handles regular expressions such that they are more compatible with the character set allowed in PCB Editor object names. Do not use this function to test patterns sent to the Skill *regexp* family of functions.

For value, match the database formats into the value string to contain the units preference if applicable for the property. If the data type of the attribute is BOOLEAN, and if it exists on the element, the string is empty. If the data type is INTEGER or REAL, the user units string, if any, is appended to the value. If the data type is one of the "units category" types, for example, ALTITUDE, PROP_DELAY, the MKS package converts the value.

All names are case insensitive.

Note: Property names may change from release to release, or may be rendered obsolete. Skill programs using property names may require modifications in future releases.

Arguments

<i>t_objectType</i>	String for PCB Editor database object type. Must be: compdef, component, drc, net, symdef, symbol, or group.
<i>t_property</i>	String name of property.
<i>t_value</i>	Option property value.

Allegro PCB and Package User Guide: SKILL Reference

Selection and Find Functions

g_regularExpression t if property value is to be treated as a regular expression,
 or nil, which is the default, to treat property value as a
 simple match.

Value Returned

t One or more *dbids* added to the select set.

nil No *dbids* added to the select set.

Example 1

```
p = axlSelectByProperty( "net" "ELECTRICAL_CONSTRAINT_SET" )  
axlAddSelectObject(p)
```

Selects all nets with an ECset property, then adds them to the current select set.

Example 2

```
p = axlSelectByProperty( "net" "ELECTRICAL_CONSTRAINT_SET" , "SPITFIRE_ADDRESS" )
```

Selects all nets with ECset property of value SPITFIRE_ADDRESS.

Example 3

```
p = axlSelectByProperty( "net" "ELECTRICAL_CONSTRAINT_SET" , "SPITFIRE*" t )
```

Selects all nets with ECset property with any value matching spitfire.

Allegro PCB and Package User Guide: SKILL Reference
Selection and Find Functions

Interactive Edit Functions

Overview

This chapter describes the basic database edit functions `axlDeleteObject` and `axlDBDeleteProp`. It also describes `axlShowObject`, which you can use to display the data about an object.

`axlDeleteObject` does not allow you to delete PCB Editor logical or parameter objects. Also, certain figure or property objects may be marked `readOnly`. `axlDeleteObject` ignores objects with that property. DRC markers created by PCB Editor are an example of `readOnly` PCB Editor figure objects. An AXL program cannot modify DRC objects directly.

AXL/SKILL Interactive Edit Functions

This section lists interactive edit functions.

axlChangeWidth

```
axlChangeWidth(  
    lo_dbid/o_dbid  
    f_newWidth  
)  
==> lo_dbid/nil
```

Description

Changes width of lines, clines and segments (arc and line).

Note: If you need to change the width of multiple lines, it is more efficient to pass them as a list of *dbids* than to call this function for each *dbid*. This function does not support change in the width of shape borders.

Arguments

lo_dbid/o_dbid Single *dbid* or list of *dbids*.

f_newWidth New width of line.

Value Returned

List of width objects or *nil* if failed.

Failures:

- *dbid* is not a cline, line or line/arc segment of a line/cline.
- Illegal option types.
- Transformed object is outside of database extents.

Example

Changes the width of a cline to 20 in current database use units

```
; ashOne is a selection utility found at  
;   <cdsroot>/pcb/examples/skill/ash-fxf/ashone.il  
dbid = ashOne()  
; pick a line, cline or segment (set find filter)  
updatedDbid = axlChangeWidth(dbid, 20.0)
```

See also `axlTransformObject`.

axlDeleteObject

```
axlDeleteObject(  
    lo_dbid  
)  
⇒ t/nil
```

Description

Deletes single or list of database objects from database.
Deletion of components deletes the symbol owner as well.
Deletion of nets is LOGIC only, and does no ripup.

Except for Nets, objects will get erased before they are deleted.
Only the Net's Ratsnests will get erased. Other parts of a Net will not get erased because there is no ripup. If a Net is in a highlighted state, it will get dehighlighted.

It will also allow deletion of the following parameter records:

artwork (films)
subclasses - subclasses must be empty and legal for deletion (e.g. can't delete PIN subclasses).

In the case of deleting parameter records, the current restriction is to only pass that single object. Do not try pass multiple parameter objects or to mix them with non-parameter objects.

Arguments

lo_dbid *dbid*, or list of *dbids* to delete from layout.

Value Returned

t Deleted one or more objects from the layout.

nil Deleted no objects from the layout.



If passed component or net dbid will delete the logic. This is different from the Allegro delete command which will delete the physical objects associated with the logic (clines/vias for nets and symbol for

components). To emulate the Allegro delete command behavior select then set objects selection using axlSetFindFilter with the equivlogic parameter passed to the ?enabled option (See example below).

Example

The following example loops on axlSelect and axlDeleteObject, deleting objects interactively selected by user. This could be dangerous because object is deleted without allowing oops (left as an exercise to the reader -- required use of axlDBStartTransaction and popup enhancement).

```
(defun DelElement ()  
    let ((mypopup)  
        "Delete selected Objects"  
        mypopup = axlUIPopupDefine(nil  
            ' ( ("Done" axlFinishEnterFun)  
                ( "Cancel" axlCancelEnterFun) ))  
        axlUIPopupSet(mypopup)  
        axlSetFindFilter(?enabled '("ALL" "EQUIVLOGIC") ?onButtons '("ALL"))  
        while( axlSelect() axlDeleteObject(axlGetSelSet()))  
        axlUIPopupSet( axlUIPopupDefine(nil nil))  
    ))
```

The following deletes the TOP artwork film record

```
p = axlGetParam("artwork:TOP")  
axlDeleteObject(p)
```

axlDBDeleteProp

```
axlDBDeleteProp(  
    lo_attach  
    lt_name  
)  
⇒ l_result/nil
```

Description

Deletes the properties listed by name, in *lt_name*, from the objects whose *dbids* are in *lo_attach*.

Arguments

<i>lo_attach</i>	List of <i>dbids</i> of objects from which properties are to be deleted. <i>lo_attach</i> may be a single <i>dbid</i> . If <i>lo_attach</i> is nil, then the property is to be deleted from the design itself.
<i>lt_name</i>	List of names of the properties to be deleted. <i>lt_name</i> may be a list of strings for several properties, or a single string, if only one property is to be deleted.

Value Returned

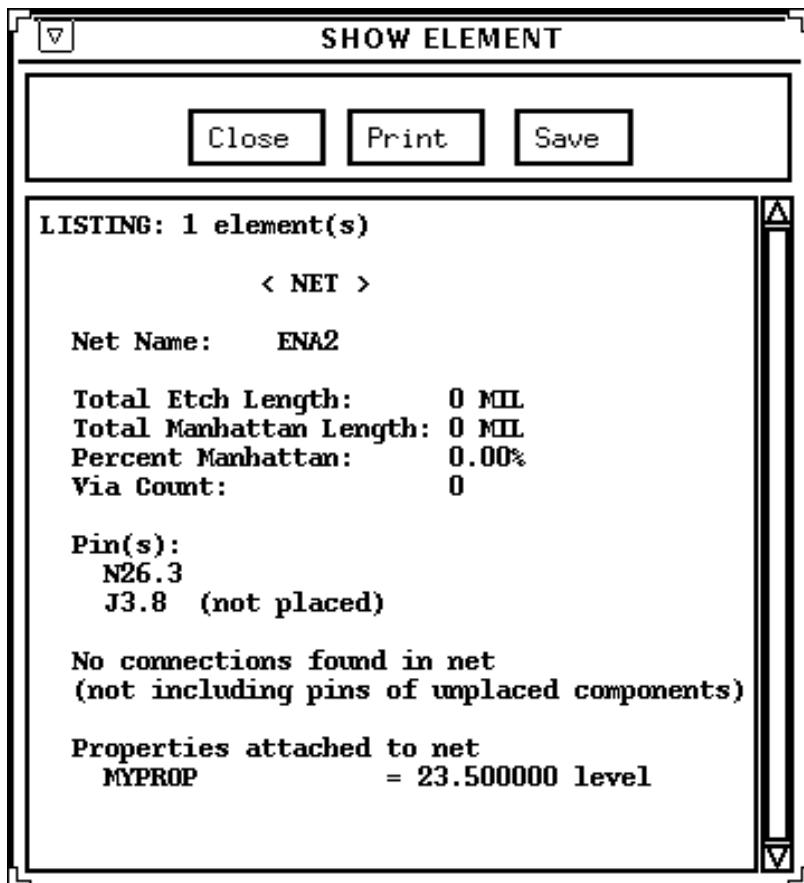
<i>l_result</i>	List. (car) list of <i>dbids</i> of members of <i>lo_attach</i> that successfully had at least one property deleted. (cadr) always nil.
nil	No properties deleted.

Example

```
axlDBCreatePropDictEntry(  
    "myprop", "real", list( "pins" "nets" "symbols"),  
    list( -50. 100), "level")  
axlClearSelSet()  
axlSetFindFilter(  
    ?enabled '("NOALL" "ALLTYPES" "NAMEFORM")  
    ?onButtons "ALLTYPES")  
axlSingleSelectName( "NET" "ENA2")  
axlDBAddProp(axlGetSelSet(), list( "MYPROP" 23.5))  
axlShowObject(axlGetSelSet())
```

First defines the string-valued property "myprop" , then adds it to the net "ena2" , then deletes the property from the net.

The following **Show Element** form shows the net with "MYPROP" attached.



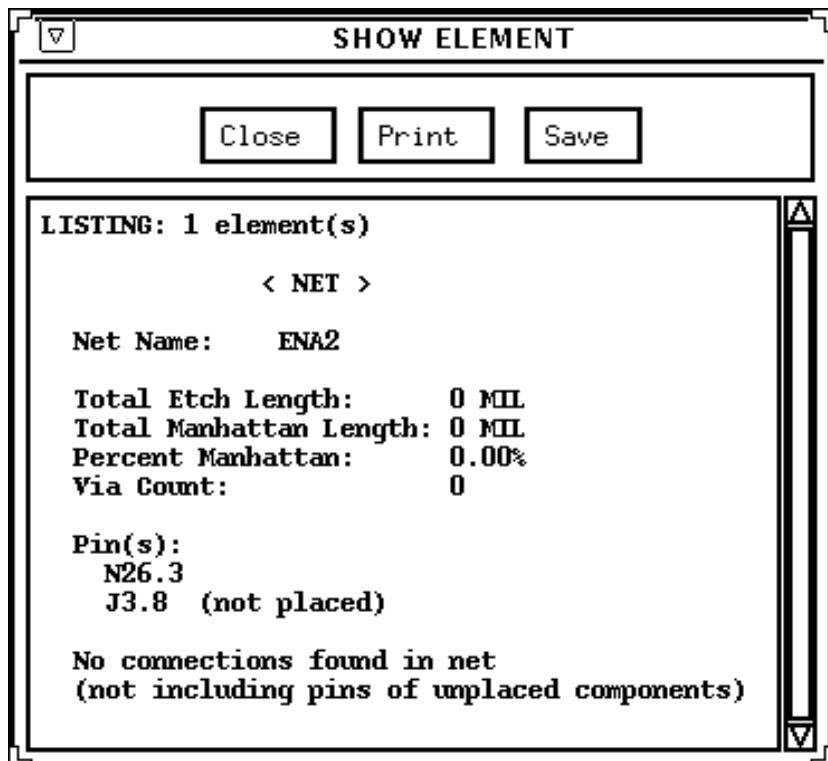
Allegro PCB and Package User Guide: SKILL Reference

Interactive Edit Functions

```
axlDBDeleteProp(axlGetSelSet() list("myprop"))
axlShowObject(axlGetSelSet())
```

Using `axlDBDeleteProp`, deletes the attached property.

The following Show Element form shows the net with *MYPROP* deleted.



axlGetLastEnterPoint

```
axlGetLastEnterPoint ( )  
⇒ l_point/nil
```

Description

Gets the last pick location from `axlEnterPoint`.

Arguments

None.

Value Returned

`axlGetLastEnterPoint` User pick from last call to `axlEnterPoint()`.

Example

Returned list for a pick: (1000.000 2000.000).

axlWindowBoxGet

```
axlWindowBoxGet(  
)  
⇒ l_bBox
```

Description

Returns the bounding box of the PCB Editor window currently viewable by the user.

Arguments

None.

Value Returned

<i>l_bBox</i>	bBox of the PCB Editor window.
---------------	--------------------------------

axlWindowBoxSet

```
axlWindowBoxSet(  
    l_bBox  
)  
⇒ l_bBox/nil
```

Description

Sets PCB Editor display to given bBox. Adjusts it according to the aspect ratio and returns the adjusted bBox.

Arguments

l_bBox bBox for display change.

Value Returned

l_bBox Adjusted bBox.

nil Invalid argument.

axlReplacePadstack

```
axlReplacePadstack (
    o_dbid/lo_dbid
    o_padstackdbid/t_padname
)
⇒ lo_dbid
```

Description

Replaces the padstack on a pin or via (or a list of them). Will not print any error messages unless you have argument errors.

The pin/via can be a list or a single *dbid*. Ignores items in the list that are not pins or vias.

The padstack can be referenced by name or a *dbid* and must be present in the PCB Editor database. Use `axlDBCreatePadStack` to obtain a *dbid*.

Returns a list of pins/vias that have had their padstacks changed. This may not be the same as your initial list as the software removes *dbids* that are not pins or vias and those items where changing the padstack would create a database error.

Note: This function will not change symbol definition pins.



Changing the padstack on a pin in the drawing editor results in an exploded pin which increases your database size and impacts refresh_symbol.

Using this function can result in disconnects and new DRC violations.

Performance Hints

To change all instances of a particular padstack, it is faster to change the padstack itself.

If you are changing many pins and vias to the same padstack, you can save time by calling this function with a list of pins/vias instead of calling it for each pin or via.

axlDeleteFillet

```
axlDeleteFillet(  
    o_dbid  
)  
⇒ t/nil
```

Description

Deletes fillet, which is a set of dangling clines with the property **FILLET**, from the **PIN**, **VIA**, or **T**.

Arguments

o_dbid *dbid* of a **PIN**, **VIA**, or **T**.

Value Returned

t Fillet deleted.

nil No fillet deleted.

axlFillet

```
axlFillet (
  o_dbid
)
⇒ t/nil
```

Description

Adds fillet between cline and pin/via, and at T. Removes and re-generates existing fillets. Fillet parameters are controlled from the **Glossing Pad and T Parameter** form.

Arguments

o_dbid *dbid* can either be a NET or CLINE.

Value Returned

t Fillet added.

nil No fillet added.

Notes

Pins, vias and Ts are not supported; use `axlDBGetConnect` on these objects to get a list of clines that connect.

For best performance, especially if fillets impact dynamic shapes, make a single call with the list of objects to be filleted.

Examples

```
fillet new MEMDATA8

axlFillet(car(axlSelectByName( "NET"  "MEM_DATA8" )) )
```

axlPurgePadstacks

```
axlPurgePadstack (      S_mode
    t/nil
)
⇒ x-cnt
```

Description

Purges unused padstacks from the database in the area controlled by *S_mode* symbol.

S_mode symbol	2nd arg = t	2nd arg = nil
'padstacks	Only purges unused derived padstacks.	Purges all unused padstacks.
'via	Purges vias not found from all via list constraints under the physical rule set and purges vias not loaded in the database, but found by looking on the disk via the PSMPATH environment variable.	Purges vias not found from all the via list constraints under the physical rule set. The nil option is NOT available from the PCB Editor user interface.



Tip
For best results, first delete the unused padstacks from the database, then purge the via lists.

Arguments

<i>S_mode</i>	'padstacks or 'via.
<i>option</i>	t - purge unused derived padstacks or nil - purge all

Value Returned

x_cnt Number of padstacks eliminated.

Examples

```
axlPurgePadstacks('padstacks nil)  
axlPurgePadstacks('via t)
```

Emulates the default PCB Editor user interface behavior.

axlShapeAutoVoid

```
axlShapeAutoVoid(  
    o_shapeId  
    [ s_options/ls_options ]  
)  
==> lo_shapeIds/nil
```

Description

Autovoids a static shape using current static shape parameters to control voiding except where options provide an override. Voiding dynamic shapes or dynamically-generated shapes is not supported.

This function produces a file, `shape.log`, as a side effect of the autovoid.

Options:

- '*noRipThermals*' - by default autovoid rips up all existing thermal ties in the shape and creates a new set, maintaining existing thermals.
- '*fragment*' - by default, if shape fragments into multiple shapes, prompts you before proceeding. If you proceed, PCB Editor allows a silent fragment. Overrides setting in static shape parameter record.
- '*noFragment*' - opposite of fragment. API fails if shape needs to be fragmented.



Do not use this function to void shapes on negative planes. Artwork does not represent inside voiding.

Arguments

o_shapeId Voidable shape.

s_options Single option symbol (see above).

ls_options List of options (see above).

Value Returned

lo_shapeId List of voided shape. Normally this is one shape unless shape is broken into multiple pieces.

nil Failed to void or illegal arguments.

See also `axlShapeDeleteVoids`.

Examples

See `<cdsroot>/share pcb/examples/skill/axlcore/ashshape.il`

```
axlShapeAutoVoid(shapeDbid '(noRipThermals fragment))
```

axlShapeChangeDynamicType

```
axlShapeChangeDynamicType(  
    o_shapeId  
    g_dynamic  
    g_msgs  
) -> o_dynShapeId/l_staticShapeId/nil
```

Description

Swaps a connectivity shape from static to dynamic or the reverse. This offers the same functionality as the PCB Editor command `shape change type`.

Notes:

- Voids in static are deleted when shape is converted to dynamic.
- Converting a dynamic shape to static can result in the loss of the original boundary since PCB Editor converts the generated shapes (on ETCH) to static shapes not boundary shapes.
- Shapes converted to static maintain voids.



If changing the type of multiple shapes or doing multiple operations on a single shape (for example, convert then raise priority) consider wrapping the code in `axlDBCloak` to batch updates.

Arguments

<code>o_shapeId</code>	Dynamic shape id or static id.
<code>g_dynamic</code>	<code>t</code> makes the shape dynamic, <code>nil</code> makes the shape static.
<code>g_msgs</code>	<code>t</code> issue error messages if failed to convert; else be silent

Value Returned

<code>nil</code>	Failure.
<code>o_dynShapeId</code>	dbid of the dynamic shape converted from static.

l_staticShapeId List of static shapes converted from dynamic shapes.

See also `axlShapeChangeDynamicType`.

Examples

See `<cdsroot>/share pcb/examples/skill/axlcore/ashshape.il`

Change to dynamic shape with messages:

```
ret = axlShapeChangeDynamicType(shape t t)
```

Change to static shape; no messages

```
ret = axlShapeChangeDynamicType(shape nil nil)
```

axlShapeDeleteVoids

```
axlShapeAutoVoid(  
    o_shapeId/o_voidId/lo_voidid  
) -> t/nil
```

Description

Lets you delete voids in a shape. Supports the following forms of arguments:

- Shape which deletes all voids in that shape
- Delete single void
- Delete list of voids

Non-voids in list of voids options are silently ignored. You can delete the voids of auto-generated shapes.

Arguments

<i>o_shapeId</i>	Given a shape; deletes all voids associated with that shape.
<i>o_voidId</i>	Deletes the given void.
<i>lo_voidid</i>	Deletes the list of voids.

Value Returned

t Deletes voids.

nil Error.

See also `axlShapeAutoVoid`.

Examples

See `<cdsroot>/share pcb/examples/skill/axlcore/ashshape.il`

Assuming you have shape dbid (`shapeId`):

Delete a single void

```
axlShapeDeleteVoids(car(p->voids))
```

Allegro PCB and Package User Guide: SKILL Reference

Interactive Edit Functions

Delete all voids in shape except first:

```
axlShapeDeleteVoids(cdr(p->voids))
```

Delete all voids in the shape:

```
axlShapeDeleteVoids(p)
```

axlShapeDynamicUpdate

```
axlShapeDynamicUpdate(  
    o_shapeDbid/nil  
    g_force  
) -> x_ood/nil
```

Description

Updates a dynamic shape, or if `nil`, all dynamic shapes are updated. This ignores the current dynamic shape mode setting of the design.

By default, only updates the shape if it is out of date unless `g_force` is `t`. In this case, it updates the shape. If `g_force` is `nil` the shape is only updated if `dbid->fillOOD` is `t`. This function supports shapes whose `dbid->shapeIsBoundary` is `t`. Updating a dynamic shape includes voiding, artwork smoothing, and thermal relief generation.

Arguments

`o_shapeDbid` dbid a dynamic shape.

`g_force` Force shape to update even if it is up to date.

Value Returned

`x_ood` If updating all returns count of all shapes that failed in updating.
If single shape returns 0; update successful, 1 otherwise.

`nil` Return if there is an error; dbid is not a dynamic shape.

Examples

Force update of one dynamic shape:

```
axlShapeDynamicUpdate(shapeId, t) -> 0
```

Update all shapes ood:

```
axlShapeDynamicUpdate(nil nil) -> 0
```

axlShapeRaisePriority

```
axlShapeRaisePriority(  
    o_shapeId  
) -> x_priority/nil
```

Description

Raises the voiding priority of a dynamic shape (*o_shapeId*) to the highest on the chosen layer. If this shape overlaps other dynamic shapes on the layer, the other shapes void away from this shape.

The priority number is relative. PCB Editor adjusts the numbers, as necessary. You should only use the priority number for comparison with other dynamic shape priority numbers.

For a dynamic shape (those on CLASS=BOUNDARY) the attribute priority reflects the current priority (for example, `dbid->priority`).



If raising priority on multiple shapes or doing multiple operations on a single shape (for example, convert; then raise priority) consider wrapping the code in `axlDBCloak` to batch updates.

Arguments

o_shapeId Dynamic shape id.

Value Returned

x_priority > 0 New priority of shape.

-1 Already at highest priority.

nil Not a dynamic shape.

See also `axlShapeChangeDynamicType`.

Example

See <cdsroot>/share pcb/examples/skill/axlcore/ashshape.il

```
axlShapeRaisePriority(shape)
```

axlShoveItems

```
list  
axlShoveItems(  
    l_itemList  
)  
⇒ t/nil
```

Description

Takes a list of *dbids* and shoves them according to the parameters set using axlShoveSetParams.

Arguments

l_itemList List of *dbids* (clines, pins, or vias) to be shoved.

Value Returned

t One or more items shoved.

nil No items shoved.

Note: Pins and vias are not shoved, but the clines around them are shoved in an attempt to eliminate any DRCs between the pin/via and the cline.

The list of *dbids* passed in does not reflect the results of the shove, as the original item may be deleted and/or replaced.

Example

```
(defun ShoveElement ()  
    axlSetFindFilter (?enabled '("CLINES" "VIAS")  
                      ?onButtons '("CLINES" "VIAS"))  
  
    axlSelect()  
    axlShoveItems (axlGetSelSet())  
)
```

Shoves an item (or items) interactively selected by the user.

axlShoveSetParams

```
axlShoveSetParams(  
    l_params  
)  
⇒ t/nil
```

Description

Sets the parameters used for shoving by the `axlShoveItems`. If you do not provide all values, the indicated default is used.

Arguments

l_params List of parameters of the form:
`(shoveMode cornerType gridded smooth oop samenet)`

ShoveMode is an integer as shown:

shoveMode	Description
0	hug preferred - Items passed in try to mold around items they are in violation with (default)
1	shove preferred - Items passed in try to shove items they are in violation with.

CornerType is an integer as shown:

cornerType	Description
90	90 degree corners.
45	45 degree corners.
0	Any angle corners.

Gridded is an integer as shown:

gridded	Description
0	Ignore grids (default)
1	Perform shoves on grid.

Smooth allows smoothing of shoved traces and is an integer as shown:

smooth	Description
0	No smoothing (default)
1	Minimal smoothing.
2	More smoothing.
3	Still more smoothing.
4	Full smoothing.

Oops allows aborting the shove of DRCs result and is an integer as shown:

oops	Description
0	<i>Oops</i> off (default)
1	<i>Oops</i> if drcs are left over.

Samenet tests for samenet violations.

Note: This results in a post-shove check for drcs that is meaningful only if you also set *oops* to the "oops if drcs" value.

samenet	Description
0	No <i>samenet</i> tests (default).
1	Enable <i>samenet</i> DRC checking.

Value Returned

t	Shove parameters set.
nil	No shove parameters set.

Example

```
(defun SetParams ()
  (let (params (shoveMode 1) (cornerType 45) (gridded 1))
    params = list(shoveMode cornerType gridded)
    axlShoveSetParams(params)
  ))
```

Sets shove parameters to shove preferred, 45 degree mode, and snap to grid.

axlTransformObject

```
axlTransformObject(  
    lo_dbid/o_dbid  
    ?move          l_deltaPoint  
    ?mirror        t/nil  
    ?angle         f_angle  
    ?origin        l_rotatePoint  
    ?allOrNone     t/nil)  
)  
⇒ lo_dbid/nil
```

Description

Moves, rotates, and/or spins one object or a list of objects. Each PCB Editor database object has a legal set of transforms (see table). If the object does not accept a transform, then that transform is silently ignored.

If multiple transformations are applied, the order used is:

1. move
2. mirror
3. rotate

If `allOrNone` flag is set, then the entire transformation fails when one object's transformation fails. By default, one object's failure does not stop the transformation on the other objects. A failure is a database failure. For example, a move that puts an object outside of the database extents is a database failure. Attempting an illegal transform is NOT a failure. If one or more objects are not transformed, there is no failure.

OBJECT	MOVE	MIRROR	ROTATE	SPIN	ORIGIN (6)	NOTES
cline	X	X	X		box	
line	X	X	X		box	
symbol	X	X	X		xy	
shape	X	X	X		box	
text	X	X	X		xy	
pin	X		X		xy	4, 5
via	X	X	X		xy	

Notes

1. If object is not listed, then it accepts no transforms.
2. If object has attached text, it also has transformation applied.
3. Mirror occurs within the same class. See mirror rules.
4. Symbol is exploded and `refresh_symbol` does not maintain transformation.
5. For Pins on a board to be transformed, the `UNFIXED_PINS` either must be present on the drawing or on the symbol owning the pin.
6. `ORIGIN` shows what rotate uses when operating on a single object without the origin option. For box, `dbid` doesn't have an origin and it uses the center of its bounding box (`dbid->bBox`). For an `xy` object that has an origin (`dbid->xy`), it rotates about this. For further discussion, see the note on angles.
7. Rotation (angle option) works as follows:
 - ❑ Positive angle results in a counter-clockwise rotation.
 - ❑ If just angle is provided, then the object is rotated about its origin point. If the `dbid` has no origin, then the center of its bounding box is used. If a list of `dbids` is provided, then the rotation always occurs about the center of the object set.
 - ❑ You can provide a rotation origin.
`(?origin l_rotatePoint)`.
 This point is then used as the rotation point.

Cautions

- The return list may be changed to show the actual set of objects that were transformed.
- Spin (rotate a list of objects about each of their centers) is not supported. Use `axlTransformObject` for each object in the list.
- If you pass a list containing a symbol and pins of the symbol, you get unexpected results.
- Line/cline segments are not supported.

Arguments

<code>lo_dbid/o_dbid</code>	Single <code>dbid</code> or a list of <code>dbids</code> .
<code>l_deltaPoint</code>	Move distance.
<code>mirror</code>	Mirror object (see table)
<code>f_angle</code>	Rotation angle.
<code>l_rotatePoint</code>	Rotation point.
<code>allOrNone</code>	If <code>t</code> and a group of objects, transform must succeed on all objects, or fail.

Value Returned

<code>lo_dbid</code>	List of transformed objects.
<code>nil</code>	Failure due to one of the following: <ul style="list-style-type: none">□ An object can't be transformed (for example, a net)□ An object is fixed or a pin does not have an <code>UNFIX_PINS</code> property.□ Illegal option types used.□ Transformed object is outside of the database extents.



For better performance when transforming a group of objects, call this function with the object group instead of passing each `dbid` individually.

Examples

l_{dbid} represents a list of database objects.

Example 1

```
axlTransformObject(ldbid, ?move '(100.0, 0.0))
```

Moves a set of objects 100 database units vertically.

Example 2

```
a xlTransformObject(dbid, ?angle 45)
```

Rotates an object about its origin 45 degrees.

Example 3

```
a xlTransformObject(dbid, ?angle 45 ?origin 100:100)
```

Rotates an object about a rotation point.

Allegro PCB and Package User Guide: SKILL Reference

Interactive Edit Functions

Database Read Functions

AXL-SKILL Database Read Functions

The chapter describes the AXL-SKILL functions that read the PCB Editor database.

axlDBGetDesign

```
axlDBGetDesign()  
)  
⇒ o_design/nil
```

Description

Returns the root design *dbid*. Use this *dbid* to get the design properties and to add properties to the design.

Note: Note that you cannot edit the root design object. AXL-SKILL edit commands ignore this *dbid*.

Arguments

None.

Value Returned

<i>o_design</i>	Root design <i>dbid</i> .
<i>nil</i>	Error occurred.

Example

```
mydesign = axlDBGetDesign()  
axlDBAddProp( mydesign, list("board_thickness", 0.350))
```

Gets the root design and sets the *BOARD_THICKNESS* property to 0.350 inches.

To verify the property has the value specified:

1. From the PCB Editor menu, select *Display–Element*.
2. From the Find Filter, select *Drawing Select*.

The **Show** window appears, listing the current properties attached to the design.

axlDBGetDrillPlating

```
axlDBGetDrillPlating  
  t_padstackname  
)  
⇒ "PLATED" / "NON_PLATED" / "OPTIONAL" / nil
```

Description

Retrieves the plating type of the padstack passed as an argument to this function.

Arguments

t_padstackname Name of padstack.

Value Returned

Plated/Nonplated/Optional Drillplating name.

nil Incorrect padstack name, or other error occurred.

axlIsDBIDType

```
axlIsDBIDType(  
    g_dbid  
)  
⇒ t/nil
```

Description

Determines if *g_dbid* is an PCB Editor database *dbid*. Returns *t* if so and *nil* otherwise.

Arguments

<i>g_dbid</i>	Variable to be checked whether a <i>dbid</i> or not.
---------------	--

Value Returned

<i>t</i>	<i>g_dbid</i> is a true PCB Editor <i>dbid</i> .
----------	--

<i>nil</i>	<i>g_dbid</i> is not a true PCB Editor <i>dbid</i> .
------------	--

Example

Defines a function based on `axlIsDBIDType` to tell whether a symbol is an PCB Editor *dbid* or not. Then creates an *r_path* (which is not an PCB Editor *dbid*, because *paths* are only temporary building structures) and uses the *r_path* to create an PCB Editor line (which is an PCB Editor *dbid*). Shows whether each is a true *dbid*.

```
defun( isItDBID (testDBID)  
      "Print whether testDBID is a true Allegro dbid"  
      if( axlIsDBIDType( testDBID)  
          then  
              println( "This is an Allegro DBID." )  
          else  
              println( "This is NOT an Allegro DBID." ) ) )  
  
mypath = axlPathStart( list(100:500))  
axlPathLine( mypath, 0.0, 200:250)  
myline = axlDBCreatePath( mypath, "etch/top" nil)  
isItDBID(mypath)  
isItDBID(caar(myline))
```

The function prints the following:

```
"This is NOT an Allegro DBID."  
"This is an Allegro DBID."
```

axlDBGetAttachedText

```
axlDBGetAttachedText(  
    o_dbid  
)  
⇒ l_dbid/nil
```

Description

Returns the list of *dbids* of text objects attached to the object whose *dbid* is *o_dbid*.

Arguments

o_dbid *dbid* of object from which attached text *dbids* are retrieved.

Value Returned

l_dbid List of the text objects attached to *o_dbid*.

nil No attached text objects.

Example

```
(defun showText ()
    "Print text of selected objects"
    mypopup = axlUIPopupDefine( nil
        (list (list "Done" 'axlFinishEnterFun)
              (list "Cancel" 'axlCancelEnterFun)))
    axlUIPopupSet( mypopup)
    axlSetFindFilter( ?enabled list("noall")
                      ?onButtons "noall")
    axlSetFindFilter( ?enabled list("symbols")
                      ?onButtons "symbols")
    axlOpenFindFilter()
    (while (axlSelect)
        progn(
            alltext =
                axlDBGetAttachedText(car(axlGetSelSet())))
            foreach(thistext alltext
                printf( "Text on this symbol is : '%s'\n",
                       thistext->text))))
    axlCloseFindFilter())
```

Lets the user pick a symbol, then prints the text attributes of each text object attached to that symbol.

Run `showText()` and pick a symbol of device type "74F74" , assigned as `refdes "T23"` .
The function prints the following:

```
Text on this symbol is : 'T23'
Text on this symbol is : '74F74'
```

axlDBGetPad

```
axlDBGetPad(  
    o_dbid  
    t_layer  
    t_type  
)  
⇒ o_pad/nil
```

Description

For the pin or via specified by *o_dbid*, gets the pad of type *t_type* associated with layer *t_layer*.

Arguments

<i>o_dbid</i>	<i>dbid</i> of the pin, via, or a padstack definition.
<i>t_layer</i>	Layer of pad to retrieve, for example, "ETCH/TOP".
<i>t_type</i>	Type of pad to retrieve: "REGULAR", "ANTI", or "THERMAL".

Value Returned

<i>o_pad</i>	<i>dbid</i> of the pad of the type associated with <i>o_dbid</i> on the layer specified.
nil	Cannot get the pad <i>dbid</i> .

Example

```
(defun showPad ()
    mypopup = axlUIPopupDefine( nil
        (list (list "Done" 'axlFinishEnterFun)
              (list "Cancel" 'axlCancelEnterFun)))
    axlUIPopupSet( mypopup)
    axlSetFindFilter( ?enabled list("noall"
        ?onButtons "noall"))
    axlSetFindFilter( ?enabled list("pins" "vias"
        ?onButtons list("pins" "vias")))
    (while axlSelect()
        progn(
            mypad = axlDBGetPad(car(axlGetSelSet()))
            "etch/top" "regular")
            printf( "Pad figure type : %s\n",
            mypad->figureName)))
```

Lets the user pick any pin or via and shows the *figureName* attribute of the selected pad.

Run `showPad()` and pick a pin with a square pad on "etch/top", then a circular pad. The function prints the following:

```
Pad figure type : SQUARE
Pad figure type : CIRCLE
```

axlDBGetPropDictEntry

```
axlDBGetPropDictEntry(  
    t_name  
)  
⇒ o_propDictEntry/nil
```

Description

Gets the property dictionary entry for the property name given by the string *t_name*.

Arguments

t_name String specifying the name of the property whose dictionary entry is to be retrieved.

Value Returned

o_propDictEntry *dbid* of the property dictionary entry for the property whose name is given by *t_name*.

nil Could not get the entry.

Example

```
myprop = axlDBGetPropDictEntry("signal_model")
```

Gets property "signal model".

```
myprop->??  
(write nil  
useCount 0  
units nil  
range nil  
objType "PropDict"  
name "SIGNAL_MODEL"  
dataType "STRING"  
readOnly t  
)
```

Dumps attributes of property "signal model".

axlDBGetProperties

```
axlDBGetProperties(  
    o_dbid  
    [lt_type]  
)  
⇒ l_result/nil
```

Description

Gets the properties attached to a specified object. Returns the properties in an assoc list, that is, a list of lists, each of which contains a name and a value. The SKILL `assoc` function can operate using this list.

Arguments

<i>o_dbid</i>	<i>dbid</i> of the object from which to get the properties.
<i>lt_type</i>	List of strings qualifying the types of properties to be retrieved from <i>o_dbid</i> . "user" means retrieve user-defined properties only. "allegro" means retrieve PCB Editor defined properties only. nil means retrieve both user and PCB Editor.

Value Returned

<i>l_result</i>	List of name-value pairs. For each name-value pair: (car) is the property name (cadr) is the property value, including units.
nil	No properties found.

Example

```
axlClearSelSet()
axlSetFindFilter(?enabled
  '("noall" "alltypes" "nameform")
  ?onButtons "alltypes")
axlSingleSelectName( "component" "y23")
myprops = axlDBGetProperties(car(axlGetSelSet( )))
print myprops
⇒((ROOM "D"
  (DFA_DEV_CLASS "DIP")
  (LEAD_DIAMETER "23 MIL")))
```

Selects the component with refdes "Y23" , gets its properties, and prints the assoc property list it returns. The properties are: *ROOM* with value "D" , *DFA_DEV_CLASS* with value "DIP" , and *LEAD_DIAMETER* with value "23 mil" .

axlDBGetDesignUnits

```
axlDBGetDesignUnits(  
)  
⇒ l_value/nil
```

Description

Returns the design units and accuracy number of the active design.

Arguments

None.

Value Returned

<i>l_value</i>	List containing the design units as a string and the accuracy number as an integer.
nil	Failed to return the design units and accuracy number of the active design.

Example

```
(axlDBGetDesignUnits)  
⇒ ("millimeters" 3)
```

The design **Drawing Parameters** form shows *User Units* as Millimeter and *Accuracy* as 3.

ax1DBRefreshId

```
ax1DBRefreshId(  
    o_dbid/nil  
)  
⇒ o_dbid/nil
```

Description

Updates the attributes of the object specified by *o_dbid*. Subsequent attribute retrieval requests access the updated information.

Note: Because of performance considerations, refreshes only the object itself. If the object being refreshed has *dbids* in any of its attributes, those *dbids* are not refreshed. For example, a net branch has *children*, a list of paths, tees, vias, pins, and shapes. If another path is added to that list of paths due to connectivity change, ax1DBRefreshId of the branch does not update the *children*. If you move a via that is a child of the branch, then doing ax1DBRefreshId of the branch and accessing the via as child of branch may yield incorrect attributes of that child (via in this case).

Arguments

<i>o_dbid</i>	SKILL list of <i>dbids</i> of the objects whose attributes are to be refreshed.
---------------	---

nil	All ids are refreshed.
-----	------------------------

Note: Refreshing all ids may cause performance problems if done indiscriminately.

Value Returned

<i>o_dbid</i>	Refreshed <i>dbid</i> .
---------------	-------------------------

nil	Could not refresh.
-----	--------------------

Example

```
axlSetFindFilter( ?enabled
                  '("noall" "alltypes"))
axlSingleSelectName("net" "sclk1")
mynet = car(axlGetSelSet())
mybranch = car(mynet->branches)
mychildren = mybranch->children
foreach( thismember mychildren
          if( (thismember->objType == "via")
              then
                  axlDeleteObject(thismember)))
axlDBRefreshId(mybranch)
⇒ t
```

Finds *net* "sclk1" , walks all members of its first branch, deleting any vias. Then refreshes the branch.

If the refresh was not done, *mybranch* would still report having vias following the operation that deleted its vias.

axlDBGetLonelyBranches

```
axlDBGetLonelyBranches(  
)  
⇒ l_dbid/nil
```

Description

Returns a list of the *standalone branch dbids* in the design. A *standalone branch* is a branch not associated with any net.

Arguments

None.

Value Returned

<i>l_dbid</i>	List of standalone branches.
nil	No standalone branches found.

Example

```
(axlDBGetLonelyBranches)  
⇒(dbid:12051156 dbid:11994768 dbid:12002292 dbid:12000892 dbid:11999396  
dbid:11996652 dbid:11996048 dbid:11994476 dbid:11992964 dbid:11991564  
dbid:11989672 dbid:11989344 dbid:12072172 dbid:11895392 dbid:11892048  
dbid:11888704 dbid:11888744 dbid:11888804 dbid:11888844 dbid:11888884  
dbid:12074948 dbid:11888984 dbid:11889064 dbid:11889204 dbid:11889224  
dbid:11889856 dbid:11890036 dbid:11890056 dbid:11890236 dbid:11890256  
dbid:11886180 dbid:12011360 dbid:11886760 dbid:11887140 dbid:11887916 )
```

Gets list of standalone branch *dbids*.

axlDBGetConnect

```
axlDBGetConnect(  
    o_dbid  
    t_full  
)  
⇒ l_result/nil
```

Description

Finds all the elements other than shapes that are connected to a given *dbid*. Input can be a PIN, VIA, T, CLINE/CARC or CLINE/CARC SEGMENT.

If *t_full* is *nil*, the function returns a list of objects connected to either end for CLINES and segments. The function returns a list of connected clines for pins, vias, or Ts.

If *t_full* is *t*, the function returns connectivity of the *dbid* including shapes. For pins, vias, or Ts, it returns full connectivity. For shapes, it returns a list of connected objects, including clines, shapes, pins, vias or Ts.

Note: You should set *t_full* to *t*. The *nil* option is only available for legacy purposes.

Arguments

<i>o_dbid</i>	A <i>dbid</i> , cline, segment, shape, pin, via or T.
<i>t_full</i>	<i>t</i> : For full connectivity of pins, vias, or Ts. <i>nil</i> : Returns connectivity excluding any connected SHAPES. Also supports segments.

Value Returned

<i>l_result</i>	List of <i>dbids</i> connected to <i>o_dbid</i> . If <i>o_dbid</i> is a CLINE or SEGMENT, then <i>l_result</i> = (list <i>list1</i> <i>list2</i>) where <i>list1</i> = <i>nil</i> or elements connected to the first end <i>list2</i> = <i>nil</i> or elements connected to the second end. For all other objects, returns a list of connections.
-----------------	---

Note: SHAPE is not supported.

<i>nil</i>	Nothing connected to <i>o_dbid</i> .
------------	--------------------------------------

axlDBGetManhattan

```
axlDBGetManhattan(  
    o_dbid_net  
)  
⇒ l_result/nil
```

Description

Given a net, calculates an etch, path, and manhattan length. The result is the same as that used by list element.

- Etch - The current length of etch. The length is 0 when there is no etch.
- Path - The etch plus remaining length. When the net is fully connected, there is no remaining, and path is equal to etch.
- Manhattan - The estimated routing length.

Note: Path is equal to manhattan when the net has no etch.

Arguments

o_dbid Net *dbid*.

Value Returned

l_result (*etchLength path manhattan*)

nil Not a net *dbid*.
 Net is out of date.
 No ratsnest.

Example

```
p = ashOne()  
axlDBGetManhattan(p)  
(2676.777 3300.0)
```

axlDBIsBondpad

```
axlDBIsBondpad(  
    o_dbid  
)  
⇒ t/nil
```

Description

Verifies whether or not the given element is a *bondpad*.

A *bondpad* (or *bondfinger*) is a “via” with the BOND_PAD property.

Arguments

o_dbid *dbid* of the element to be checked.

Value Returned

t *o_dbid* is a bondpad.

nil *o_dbid* is not a bondpad.

axlDBIsBondwire

```
axlDBIsBondwire(  
    rd_dbid  
)  
⇒ t/nil
```

Description

Verifies whether or not the given element is a *bonding wire*.

A *bonding wire* is a “via” with either the “beginning” or “ending” layer type set to the BondingWire property.

Arguments

rd_dbid *dbid* of the element to check.

Value Returned

t *rd_dbid* is a bonding wire.

nil *rd_dbid* is not a bonding wire.

axlDBIsDiePad

```
axlDBIsDiePad(  
    rd_dbid  
)  
⇒ t/nil
```

Description

Verifies whether or not the given element is a *die pad*.

A *die pad* is a pin with a component class of IC.

Arguments

rd_dbid *dbid* of the element to check.

Value Returned

t *rd_dbid* is a die pad.

nil *rd_dbid* is not a die pad.

axlDBIsFixed

```
axlDBIsFixed(  
    o_dbid  
    [g_showMessage ]  
)  
⇒ nil or [dbid of 1st element that makes the item fixed]
```

Description

Verifies whether or not the database object is fixed.

An object can be fixed by the following:

- Object has the `FIXED` property or has parents with the `FIXED` property.
- Object is a symbol with test points and the `FIXED` test point flag is set.
- Object is a symbol and has one or more children with the `FIXED` property.

Returns the first item found that caused the element to be fixed.

Arguments

o_dbid *dbid* of the element to check.

g_showMessage Use `t` to have PCB Editor display the message if the item is fixed or `nil` to have no message display.

Value Returned

dbid *dbid* of the element causing the object to be fixed.

nil Object not fixed.

Example

```
p = axlSelectByname( "SYMBOL"  "U1" )  
ret = axlDBIsFixed(p)
```

axlDBIsPackagePin

```
axlDBIsPackagePin(  
    rd_dbid  
)  
⇒ t/nil
```

Description

Verifies whether or not the given element is a *package pin*.

A *package pin* is a pin with a component class of `IO`.

Arguments

`rd_dbid` *dbid* of element to check.

Value Returned

`t` *rd_dbid* is a package pin.

`nil` *rd_dbid* is not a package pin.

axlDBIsPlatingbarPin

```
axlDBIsPlatingbarPin(  
    rd_dbid  
)  
⇒ t/nil
```

Description

Verifies whether or not the given element is a *plating bar pin*.

A *plating bar pin* is a pin with a component class of DISCRETE or PLATING_BAR.

Arguments

rd_dbid *dbid* of the element to check.

Value Returned

t *rd_dbid* is a plating bar pin.

nil *rd_dbid* is not a plating bar pin.

axlGetModuleInstanceDefinition

```
axlGetModuleInstanceDefinition(  
    o_modinst  
)  
⇒ t_moddef/nil
```

Description

AXL interface to the C function that returns the name of the module definition used to create the module instance.

Arguments

o_modinst AXL *dbid* of the module instance (the *dbid* returned by axlDBCreateModuleInstance.)

Value Returned

t_moddef String containing the name of the module definition.

nil Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))  
axlSingleSelectName("GROUP" "inst")  
modinst = car(axlGetSelSet())  
axlGetModuleInstanceDefinition(modinst)  
= "mod"
```

Gets the definition of a module instance named *inst*.

axlGetModuleInstanceLocation

```
axlGetModuleInstanceLocation(  
    o_modinst  
)  
⇒ l_loc/nil
```

Description

AXL interface to the C function that gets the current location of the module instance in the design.

Arguments

o_modinst AXL *dbid* of the module instance (the *dbid* returned by axlDBCreateModuleInstance.)

Value Returned

l_loc List of data describing the location of the module instance. The first element is a list containing the origin in the form (x y). The second element is the rotation.

nil Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))  
axlSingleSelectName("GROUP" "inst")  
modinst = car(axlGetSelSet())  
axlGetModuleInstanceLocation(modinst)  
= ((500 1500) 0)
```

Gets the location of a module instance named *inst*.

axlGetModuleInstanceLogicMethod

```
axlGetModuleInstanceMethod(  
    o_modinst  
)  
⇒ i_logic/nil
```

Description

AXL interface to the C function that determines the logic method used by the module instance.

Arguments

o_modinst AXL *dbid* of the module instance (the *dbid* returned by `axlDBCreateModuleInstance()`.)

Value Returned

<i>i_logic</i>	Value of the logic method flag for the module instance. Legal values are: 0 - no logic 1 - logic from schematic 2 - logic from module definition
nil	Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))  
axlSingleSelectName("GROUP" "inst")  
modinst = car(axlGetSelSet())  
axlGetModuleInstanceLogicMethod(modinst)  
= 2
```

Gets the logic method of a module instance named *inst*.

axlGetModuleInstanceNetExceptions

```
axlGetModuleInstanceNetExceptions(  
    o_modinst  
)  
⇒ l_nets/nil
```

Description

AXL interface to the C function that gets the net exception of the module instance in the design.

Arguments

o_modinst AXL *dbid* of the module instance (the *dbid* returned by *axlDBCreateModuleInstance*.)

Value Returned

l_nets List of names of the nets that are treated as exceptions in the module instance.

nil Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))  
axlSingleSelectName("GROUP" "inst")  
modinst = car(axlGetSelSet())  
axlGetModuleInstanceNetExceptions(modinst)  
= ("GND" "+5")
```

Gets the list of net exceptions of a module instance named *inst*.

axlIsDummyNet

```
axlIsDummyNet(  
    net_dbid  
)  
⇒ t/nil
```

Description

Determines if a given net is a Dummy net.

Arguments

net_dbid Net database object.

Value Returned

t *net_dbid* is a Dummy Net.

nil *net_dbid* is not a Dummy Net.

axlIsLayerNegative

```
axlIsLayerNegative(  
  t_layerName  
)  
⇒ t/nil
```

Description

Determines whether or not the given plane layer is negative.

Arguments

t_layerName Name of the conductor layer to check.

Value Returned

t Active layer is negative.

nil Active layer is not negative or is not an `ETCH` layer.

axlIsPinUnused

```
axlIsPinUnused(  
    pin_dbid  
)  
⇒ t/nil
```

Description

Determines if a given pin is unused.

Arguments

<i>pin_dbid</i>	Pin database object.
-----------------	----------------------

Value Returned

t	Pin is unused.
---	----------------

nil	Pin is used.
-----	--------------

axlIsitFill

```
axlIsitFill(  
    t_layer  
)  
⇒ t/nil
```

Description

Determines if fill shape is allowed for a given class subclass.

Arguments

t_layer Layer name, for example, ETCH/TOP.

Value Returned

t Fill shape is allowed.

nil Fill shape is not allowed.

axlOK2Void

```
axlOK2Void(  
    t_layer  
)  
⇒ t/nil
```

Description

Determines if voids are allowed for a given *class/subclass*.

Arguments

t_layer Layer name, for example, ETCH/TOP.

Value Returned

t Voids are allowed.

nil Voids are not allowed.

axlDBAssignNet

```
axlDBAssignNet(  
    o_object/lo_object  
    o_net/t_net  
    [g_riput]  
)  
⇒ t/nil
```

Description

Assigns an object or a list of objects to a new net. Supports pins and shapes.

Arguments

<i>o_object</i>	<i>dbid</i> , or list of <i>dbids</i> of objects to change net.
<i>o_net</i>	<i>dbid</i> of destination net, or <i>nil</i> if assigning to a dummy net.
<i>t_net</i>	Net name.
<i>g_riput</i>	<i>t</i> = ripup clines connected to modified objects. <i>nil</i> = do not ripup clines connected to modified objects. The default is <i>nil</i> .

Value Returned

<i>t</i>	At least one object changed net.
<i>nil</i>	No object changed net.

Example

```
pin_id->net->name  
=> ""  
net_id->name  
=> "GND"  
axlDBAssignNet(pin_id net_id t)  
=> t  
pin_id->net->name  
=> "GND"
```

ax1DBDynamicShapes

```
ax1DBDynamicShapes(  
    g_value  
)  
⇒ x_count
```

Description

Queries and updates dynamic shapes. When *g_value* is *t*, updates all out of date dynamic shapes on the board regardless of the dynamic shape updating setting in the **Drawing Options** dialog. When *g_value* is *nil*, returns a count of out of date shapes.

Arguments

<i>g_value</i>	<i>t</i> = update dynamic shapes <i>nil</i> = return count of out of date shapes
----------------	---

Value Returned

<i>x_count</i>	Count of out of date shapes. If updating shapes, <i>x_count</i> is the number of out of date shapes before the update.
----------------	--

axlDBGetShapes

```
axlDBGetShapes(  
    t_layer  
)  
⇒ l_dbid/nil
```

Description

Provides quick access to shapes without access to visibility or find settings.

Arguments

t_layer Layer name
 nil = all layers
 <class> = all subclasses of the class
 <class>/<subclass> = specified layer

Value Returned

<i>l_dbid</i>	List of shapes.
<i>nil</i>	Incorrect argument.

Example 1

```
axlDBGetShapes(nil)
```

Returns all shapes on the design.

Example 2

```
axlDBGetShapes( "BOUNDARY" )
```

Returns all shapes on the BOUNDARY layer.

Example 3

```
axlDBGetShapes( "ETCH/GND" )
```

Returns all shapes on ETCH GND.

Example 4

```
axlDBGetShapes( "ROUTE KEEPOUT" )
```

Returns all shapes on ROUTE KEEPOUT.

axlDBIsBondingWireLayer

```
axlDBIsBondingWireLayer(  
    t_layerName  
)  
⇒ t/nil
```

Description

Verifies if a layer is a bonding layer. This means that attribute of the “paramLayer” parameter *dbid* called “type” has a value of “BONDING _WIRE”.

This is normally used in the APD product.

Arguments

t_layerName Layer name, for example, "CONDUCTOR/<subclass>"
Note: CONDUCTOR is ETCH in PCB Editor.

Value Returned

t	Layer is a bonding layer.
nil	Layer is not a bonding layer.

Example

```
axlDBIsBondingWireLayer( "CONDUCTOR/TOP_COND" ) -> nil
```

ax1DBTextBlockCompact

```
ax1DBTextBlockCompact(  
    t/nil  
)  
⇒ x_unusedBlocks
```

Description

Reports and/or compresses unused database text blocks. If compacting text blocks, it always updates database text to reflect the new text block numbers.

The database, even if new, must have at least one text block.

Note: You must force a *dbid* refresh on any text parameters and text type *dbids* in order for them to reflect the new numbering.

Arguments

t	Compact the text blocks.
nil	Report the number of text blocks that can be eliminated from the database.

Value Returned

x_unusedBlocks Count of text blocks that are unused.

Example

```
unused = ax1DBTextBlockCompact(nil)  
printf("This database has %d unused text blocks\n" unused)
```

PCB Editor Interface Functions

Overview

This chapter describes the AXL/SKILL functions that give access to the PCB Editor interface. These include display control, cursor setup, and soliciting user input, such as text and mouse picks.

AXL-SKILL Interface Function Examples

This section gives examples of the following:

- Dynamic cursor functions used with the `axlEnter` functions
- `axlCancelEnterFun` and `axlFinishEnterFun` used with the popup functions in a command looping on the `axlEnterPath` command
- `axlHighlightObject` and `axlDehighlightObject`

Dynamic Cursor Examples

You use the AXL-SKILL dynamic cursor functions to build up and display PCB Editor database objects during interactive commands. Using dynamic cursor shows the effects of a command in use. For example, you can display a symbol and the etch lines connected to it, constantly showing where they would be in the drawing if the user clicked at their current position.

The two examples that follow show how to set up the dynamic cursor:

- A package symbol image with pins connected to other etch, with rubberband lines from its connected pins to the points where they had originally connected
- A package symbol image dynamically rotating enabling you to select an angle of rotation

Both examples use the `axlPath` functions described in [Chapter 3, “Database Create Functions.”](#) and the `axlAddSimpleXXXDynamics` functions described in this chapter.

Example 1: Dynamic Rubberband

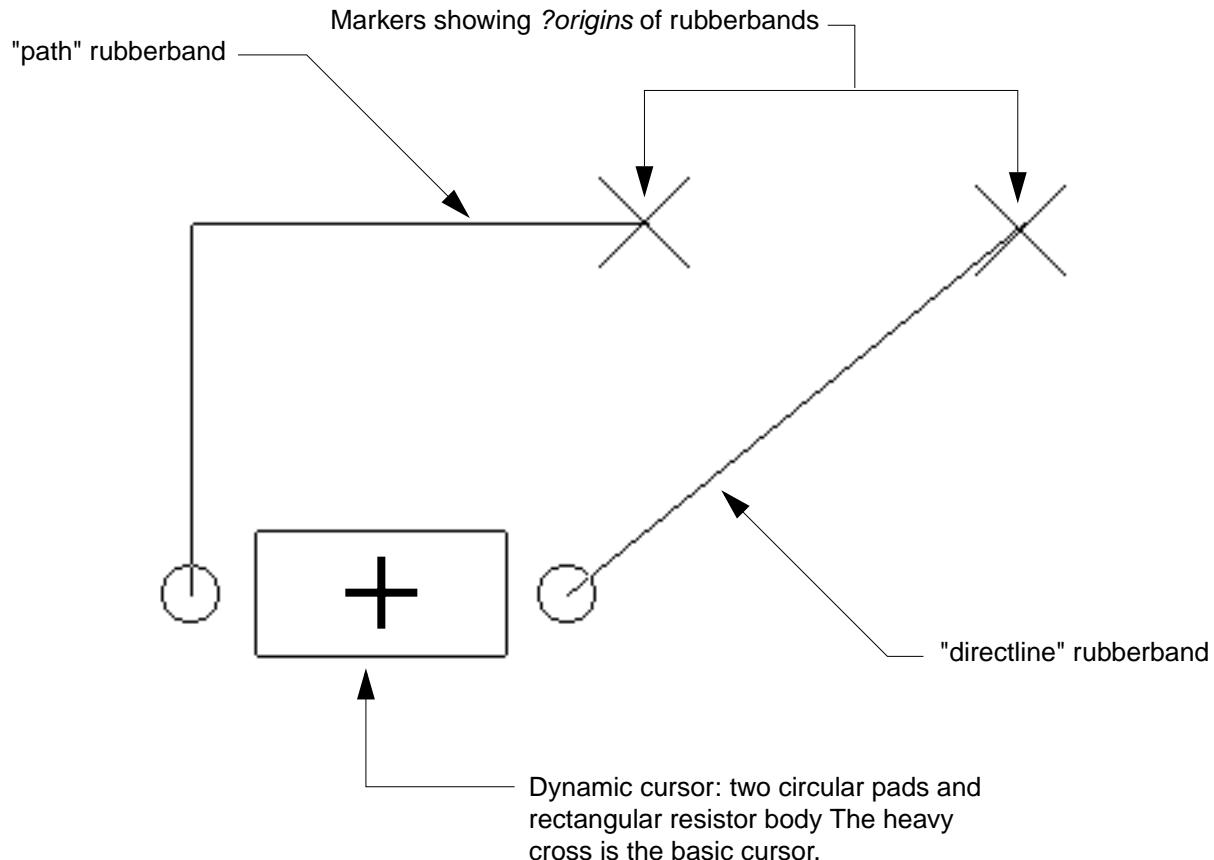
```
axlClearDynamics() ; Clean out any existing cursor data  
  
; Create cross markers to show rubberband origins:  
axlDBCreateLine(list(9150:4450 9050:4550) 0.  
                  "board geometry/dimension")  
axlDBCreateLine(list(9150:4550 9050:4450) 0.  
                  "board geometry/dimension")  
axlDBCreateLine(list(8550:4450 8450:4550) 0.  
                  "board geometry/dimension")  
axlDBCreateLine(list(8550:4550 8450:4450) 0.  
                  "board geometry/dimension")  
  
mypath = axlPathStart(list( -350:0)) ; Start circular pad  
axlPathArcCenter(mypath, 0., -350:0, nil, -300:0)  
  
; Load the first pad into the dynamic cursor buffer  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
mypath = axlPathStart(list( 350:0)) ; Start circular pad  
axlPathArcCenter(mypath, 0., 350:0, nil, 300:0)  
  
; Load the other pad into the dynamic cursor buffer  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
mypath = axlPathStart( ; Start resistor body outline  
list( -200:-100 200:-100 200:100 -200:100 -200:-100))  
  
; Load the resistor body outline in the dynamic cursor buf  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
; Load a "path" rubberband to the first pad  
axlAddSimpleRbandDynamics(8500:4500 "path"  
                           ?origin 8500:4500 ?var_point -300:0)  
  
; Load a "directline" rubberband to the second pad  
axlAddSimpleRbandDynamics(9100:4500 "directline"  
                           ?origin 9100:4500 ?var_point 300:0)  
;  
mypoint = axlEnterPoint() ; Ask user for point
```

Loads two circular pads, the outline of a resistor, and rubberband connections from its pins (one with a "path" rubberband, the other a "directline" rubberband) into the dynamic cursor buffer.

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor Interface Functions

The following illustration shows the cursor in a typical position as `ax1EnterPoint` waits for selection of a point.



Example 2: Dynamic Cursor Rotation

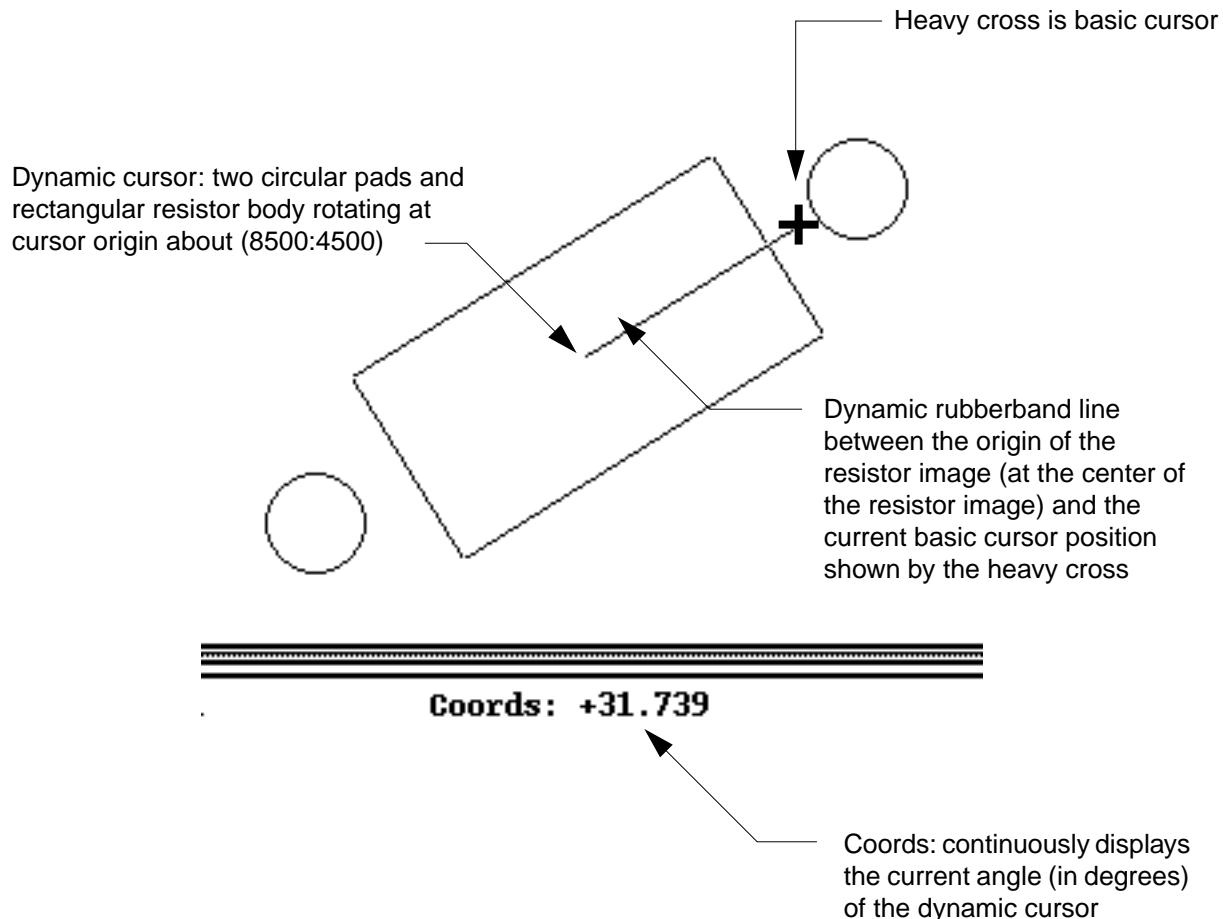
```
axlClearDynamics() ; Clean out any existing cursor data  
  
mypath = axlPathStart(list( -350:0)) ; Start circular pad  
axlPathArcCenter(mypath, 0., -350:0, nil, -300:0)  
  
; Load the first pad into the dynamic cursor buffer  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
mypath = axlPathStart(list( 350:0)) ; Start circular pad  
axlPathArcCenter(mypath, 0., 350:0, nil, 300:0)  
  
; Load the other pad into the dynamic cursor buffer  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
mypath = axlPathStart( ; Start resistor body outline  
list( -200:-100 200:-100 200:100 -200:100 -200:-100))  
  
; Load the resistor body outline in the dynamic cursor buf  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
; Ask user to pick angle of rotation about (8500:4500):  
axlEnterAngle(8500:4500)
```

Loads two circular pads, the outline of a resistor, and rubberband connections from its pins, one with a "path" rubberband, the other a "directline" rubberband into the dynamic cursor buffer.

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor Interface Functions

The following illustration shows the dynamically rotating cursor in a typical position as `axlEnterAngle` waits for a user-selected point.



Enter Function Example

You use the AXL-SKILL `axlCancelEnterFun` and `axlFinishEnterFun` functions when you create an interactive command that loops on input, providing the option to end the command.

```
(defun axlMyCancel ()
  axlClearDynamics()
  axlCancelEnterFun()
  axlUIPopupSet(nil))

(defun axlMyDone ()
  axlClearDynamics()
  axlFinishEnterFun()
  axlUIPopupSet(nil))

mypopup = axlUIPopupDefine( nil
  (list (list "MyCancel" 'axlMyCancel)
    (list "MyDone" 'axlMyDone)))
axlUIPopupSet( mypopup)
; Clear the dynamic buffer
axlClearDynamics()
; Clear mypath to nil, then loop gathering user picks:
mypath = nil
while( (mypath = axlEnterPath(?lastPath mypath))
  progn(
    axlDBCreatePath(mypath, "etch/top")))
```

The Enter Function example does the following:

1. Defines the functions `axlMyCancel` and `axlMyDone`.
2. Defines a pop-up with those functions as the callbacks for user selections *Cancel* and *Done* from the pop-up.
3. Loops on the function `axlEnterPath` gathering user input to create a multi-segment line on "etch/top".

Selecting *Cancel* or *Done* from the pop-up ends the command.

You gather one user-selected point and extend the database path by that selection each time through the *while* loop. Selecting *Done* from the pop-up terminates the loop. Selecting *Cancel* at any time cancels. Segments added become permanent in the database when the loop ends.

axlHighlightObject and axlDehighlightObject Examples

You use the AXL-SKILL `axlHighlightObject` and `axlDehighlightObject` functions to highlight database elements during interactive commands.

Example 1

```
(defun highlightLoop ()
  mypopup = axlUIPopupDefine( nil
    (list (list "Done" 'axlFinishEnterFun)
          (list "Cancel" 'axlCancelEnterFun)))
  axlUIPopupSet( mypopup)
  axlSetFindFilter( ?enabled '("noall" "alltypes" "nameform")
                    ?onButtons "alltypes")
  (while (axlSelect)
    progn(
      axlHighlightObject( axlGetSelSet())
      ; Just a dummy delay to see what happens
      sum = 0
      for( i 1 10000 sum = sum + i)
      axlDehighlightObject( axlGetSelSet()))))
```

Example 1 does the following:

1. Defines the function `highlightLoop`.
2. Defines a popup with `axlFinishEnterFun` and `axlCancelEnterFun` as the callbacks for user selections *Done* and *Cancel* from the pop-up.
3. Loops on the function `axlSelect` gathering user selections to highlight.
4. Waits in a simple delay loop, then dehighlights.

Selecting *Cancel* or *Done* from the pop-up ends the command.

Example 2

```
axlDBControl('highlightColor 4)
axlHighlightObject(axlGetSelSet() t)
```

Permanently highlights an object using color 4.

PCB Editor Interface Functions

This section lists PCB Editor interface functions.

axlClearDynamics

```
axlClearDynamics()  
)  
⇒ t
```

Description

Clears the dynamic cursor buffer. Call this function each time before you start setting up rubberband and dynamic cursor graphics.

Arguments

None.

Value Returned

t	Always returns t.
---	-------------------

Example

See dynamic cursor examples in the section [AXL-SKILL Interface Function Examples](#) on page 331.

axlAddSimpleRbandDynamics

```
axlAddSimpleRbandDynamics(
```

```
    l_fixed_point  
    t_type  
    ?origin      l_origin  
    ?var_point   l_var_point  
    ?lastPath    r_lastPath
```

```
)
```

```
⇒ t/nil
```

Description

Loads *rband* (dynamic rubberband) data incrementally into the dynamic rubberband buffer. Adds this data to any already in the rubberband buffer. Call this multiple times to build up data incrementally in the rubberband buffer.

Arguments

<i>l_fixed_point</i>	Anchor point from which the dynamic rubberband stretches. Rubberband cursor stretches dynamically from <i>fixed_point</i> to current position of the cursor, as moved by the user. The next argument, <i>t_type</i> , specifies the shape of the rubberband—part of a path, direct, z-line (a combination of horizontal and vertical), arc, circle, or box.
<i>t_type</i>	String specifying type of dynamic rubberband to be drawn. Can be one of the following: path, directline, horizline, vertline, arc, circle, or box.
<i>l_origin</i>	Cursor origin.
<i>l_var_point</i>	Variable point for <i>rband</i> .
<i>r_lastPath</i>	Previous path structure. Needed to calculate tangent point if rubberbanding starts at the end of an existing path.

Value Returned

t Successfully added data.

nil No data added.

Example

See dynamic cursor examples in the section [AXL-SKILL Interface Function Examples](#) on page 331.

axlAddSimpleMoveDynamics

```
axlAddSimpleMoveDynamics(  
    l_origin  
    r_path  
    t_type  
    ?ref_point      l_ref_point  
)  
⇒ t/nil
```

Description

Loads data incrementally into the dynamic cursor buffer. Draws path structure argument in the cursor buffer, and displays it as part of the cursor. Loading incrementally means this function adds the given data to any already in the cursor buffer.

Arguments

<i>l_origin</i>	Cursor origin.
<i>r_path</i>	Path structure containing display objects.
<i>t_type</i>	String specifying type of path: either <code>path</code> or <code>box</code> . Note that <code>path</code> includes lines and arcs. For this command, circles are a type of arc.
<i>l_ref_point</i>	Object rotation reference point.

Value Returned

<i>t</i>	Data added.
<i>nil</i>	No data added.

Example

See dynamic cursor examples in the section [AXL-SKILL Interface Function Examples](#) on page 331.

axlEnterPoint

```
axlEnterPoint(  
    ?prompts      l_prompts  
    ?points       l_points  
    ?gridSnap     g_gridSnap  
)  
⇒ l_point/nil
```

Description

Prompts for and receives user-selected point. Returns the point data to the calling function.

Arguments

<i>l_prompts</i>	List containing one prompt message to display.
<i>l_points</i>	List of points. Returns one of these as the return value. <i>l_point</i> 's only use is, if passed a point, to immediately return with the point snapped to the nearest grid.
<i>g_gridSnap</i>	Flag to function: t means snap the point according to the current grid.

Value Returned

<i>l_point</i>	List of coordinates, if entered. If selected, this is a list of one point.
<i>nil</i>	User did not select a point.

Example

See Example 1 in the section [AXL-SKILL Interface Function Examples](#) on page 331.

axlEnterString

```
axlEnterString(  
    ?prompts      l_prompts  
)  
⇒ t_string/nil
```

Description

Displays a dialog box that requires first entering a string, and then pressing *Return* on the keyboard or clicking *OK* or *Cancel*. Default prompt in the dialog box is "Enter String." You can supply a prompt string with the `?prompts` keyword. The function returns the string entered, if any. Otherwise it returns `nil`.

Note: This function is a blocker. PCB Editor will not respond to any user input until the data requested by the dialog box is provided.

Arguments

<code>l_prompts</code>	List containing one prompt message. Displays only the first string if the list contains more than one string.
------------------------	---

Value Returned

<code>t_string</code>	String entered.
-----------------------	-----------------

<code>nil</code>	No string entered, dialog box dismissed by clicking <i>Cancel</i> , or the command failed.
------------------	--

Example

```
user_name = axlEnterString(  
    ?prompts list("Please enter your name:"))  
⇒ "user name"
```

Prompts for name and collects the response in `user_name`.

Typing the name, then pressing the *Return* key returns the string entered:

axlEnterAngle

```
axlEnterAngle(  
    origin  
    ?prompts      l_prompts  
    ?refPoint     l_refPoint  
    ?angle        f_angle  
    ?lockAngle   g_lockAngle  
)  
⇒ f_angle/nil
```

Description

Optionally prompts the user. Returns the angle value entered.

Arguments

<i>origin</i>	Fixed point where two lines making up the angle meet.
<i>l_prompts</i>	List containing one prompt message.
<i>l_refPoint</i>	End point of a line from the <i>origin</i> that acts as the fixed line of the angle.
<i>f_angle</i>	Angle value in. If non- <i>nil</i> , does not prompt for a user-selected point.
<i>g_lockAngle</i>	Initial lock angle for dynamic rotation.

Value Returned

<i>f_angle</i>	Selected angle expressed in degrees.
<i>nil</i>	No angle selected.

Example

See Example 1 in the section [AXL-SKILL Interface Function Examples](#) on page 331.

axlCancelEnterFun

```
axlCancelEnterFun(  
)  
⇒ t/nil
```

Description

Terminates the wait for a user-selected point. Waiting function returns no data.

Arguments

None.

Value Returned

t	Terminates wait for user-selected point. Cancel succeeds.
nil	Fails to terminate wait for user-selected point.

Example

See the [Enter Function Example](#) on page 336.

axlFinishEnterFun

```
axlFinishEnterFun(  
)  
⇒ t/nil
```

Description

Terminates the wait for a user-selected point. Waiting function returns no data. For a one-point function (for example, `axlEnterPoint`) behaves the same as `axlCancelEnterFun`.

Arguments

None.

Value Returned

t	Terminates wait for a user-selected point.
nil	Fails to terminate wait for a user-selected point.

Example

See the [Enter Function Example](#) on page 336.

axlGetDynamicsSegs

```
axlGetDynamicsSegs (
    l_point1
    l_point2
    r_lastPath/nil
) ->
```

Description

Normally used with dynamics to calculate arc tangency of two picks to a current *r_path*. Passed coordinates may be modified to preserve tangency. Depends on the current line lock state that you set or axlSetLineLock.

Arguments

<i>point1</i>	First pick before dynamics started.
<i>point2</i>	Second pick, after dynamics completes.
<i>lastPath</i>	Previous path to use for tangency calculations. Can pass nil if not applicable.

Value Returned

l_pointList
nil

See also axlAddSimpleRbandDynamics, axlMakeDynamicsPath, axlSetLineLock.

Example

```
q = axlGetDynamicsSegs(10:10 100:100 nil)
-> (((10.0 10.0) (100.0 100.0) nil))
```

axlEnterBox

```
axlEnterBox(  
    ?prompts      l_prompts  
    ?points       l_points  
)  
⇒ l_box/nil
```

Description

Takes two points that define a box and returns them in *l_box*. Optionally prompts the user, if *l_prompts* contains no more than two strings. If *l_points* is *nil*, prompts for two points. If *l_points* contains one point, prompts only for the second point. If *l_points* contains both points, simply returns them as *l_box*.

Arguments

<i>l_prompts</i>	List that should contain two prompt messages. If list is <i>nil</i> , uses default PCB Editor prompts for soliciting a box. ("Enter first point of box" and "Enter second point of box") If list contains two strings, the first string prompts for the first point, and the second string prompts for the second point. If the list has only one string, the string prompts for both the first and the second points.
<i>l_points</i>	List of none, one, or two points. Solicits missing points interactively using the prompts given in <i>l_prompts</i> in order.

Value Returned

<i>l_box</i>	List of the lower left and upper right coordinates of the box.
<i>nil</i>	Failed to get box data.

Example

```
axlDBCreateRectangle(  
    axlEnterBox(?prompts  
        list("First rectangle point, please..."  
            "Second rectangle point, please..."))  
        t "etch/top")  
    ⇒ (dbid:12134523 nil)
```

Asks for box input to create a filled rectangle on layer "etch/top".

axlEnterPath

```
axlEnterPath(  
    ?prompts      l_prompts  
    ?points       l_points  
    ?lastPath     r_path  
)  
⇒ r_path/nil
```

Description

Gets the start point and subsequent points for a path, interactively with optional prompting, or from the optional argument *l_points*. Sets the start point to the first value of *l_points*, if any, and the second point to the second value, if any. If *r_path* is given, connects the dynamic rubberband to its most recent segment. Use axlEnterPath recursively to build up the coordinates of a path interactively.

Arguments

<i>l_prompts</i>	List containing one prompt message to display.
<i>l_points</i>	List of none, one, or two coordinates to be used as input to axlEnterPath.
<i>r_path</i>	The previously gathered part of the path. Used to calculate the tangent point for the dynamic cursor.

Value Returned

<i>r_path</i>	Path containing segments constructed from the combined points in <i>l_points</i> and the interactive input to axlEnterPath.
<i>nil</i>	Failed to get points.

Example

See the [Enter Function Example](#) on page 336.

axlHighlightObject

```
axlHighlightObject(  
    [lo_dbid]  
    [g_permHighlight]  
)  
⇒ t/nil
```

Description

Highlights the figures whose *dbids* are in *lo_dbid*.

Fewer objects support permanent highlighting than support temporary highlighting.

Note: Setting `axlDebug(t)` enables additional informational messages.

Arguments

od_dbid List of the *dbids* of figures to be highlighted.

g_permHighlight Distinguishes temporary highlighting from permanent highlighting using color.

t - use PERM highlight color
nil - use TEMP highlight color

The default is `nil`.

Value Returned

`t` Highlighted at least one figure.

`nil` Highlighted no figures due to invalid *dbids* or objects already being highlighted.

Examples

You can use the AXL-SKILL `axlHighlightObject` and `axlDehighlightObject` functions to highlight database elements during interactive commands.

This example does the following:

- a. Defines the function `highlightLoop`.
- b. Loops on the function `axlSelect` gathering user selections to highlight.
- c. Waits in a simple delay loop, then dehighlights.

You can stop the command at any time by selecting *Cancel* or *Done* from the pop-up.

```
(defun highlightLoop ()
  mypopup = axlUIPopupDefine( nil
    (list (list "Done" 'axlFinishEnterFun)
          (list "Cancel" 'axlCancelEnterFun)))
  axlUIPopupSet( mypopup)
  axlSetFindFilter( ?enabled '("noall" "alltypes" "nameform")
                    ?onButtons "alltypes")
  (while (axlSelect)
    progn(
      axlHighlightObject( axlGetSelSet())
      ; Just a dummy delay to see what happens
      sum = 0
      for( i 1 10000 sum = sum + i)
      axlDehighlightObject( axlGetSelSet()))))
```

This example permanently highlights an object using color 4:

```
axlDBControl('highlightColor 4)
axlHighlightObject(axlGetSelSet() t)
```

Also see the [axlHighlightObject and axlDehighlightObject Examples](#) on page 337.

axlDehighlightObject

```
axlDehighlightObject(  
    [lo_dbid]  
    [g_permHighlight]  
)  
⇒ t/nil
```

Description

Dehighlights the figures whose *dbids* are in *lo_dbid*.

Arguments

lo_dbid List of *dbids* of figures to be dehighlighted.

g_permHighlight Distinguishes temporary highlighting from permanent highlighting using color.

t - use PERM highlight color
nil - use TEMP highlight color

The default is nil.

Value Returned

t Dehighlighted at least one figure.

nil Failed to dehighlight any figures.

Example

See [axlHighlightObject](#) on page 351 for examples.

axIMiniStatusLoad

```
axIMiniStatusLoad (
    s_formHandle
    t_formFile
    g_formAction
    [g_StringOption]
    [t_restrict]
)
⇒ r_form/nil
```

Description

Loads the Ministatus form with the form file provided in this call. Replaces the current Ministatus form contents. This function is a special case of `axlForms`. See [Chapter 11, “Form Interface Functions,”](#) for details on how AXL forms work.

When the command is finished, PCB Editor restores the Ministatus contents to the default values. Once the form is opened, you use normal `axlForm` functions to set or retrieve fields.

You typically use this to write a command requiring user interaction such as “swap component.”

Two reserved field names are available:

```
class -- enumerated list of CLASS layers
subclass -- enumerated list of SUBCLASS layers for the current active class.
```

If you make use of these fields use support changing the active class and subclass you also get (for free) color swatch support. The Form file fragment shown below can be added to your ministatus form file to get that support. The "subcolor" field is optional. You should adjust the position (FLOC) of the fields to suite your form layout.

Note: Using these reserved names will also cause `axlGetActiveLayer` to update when user changes the layer.

```
TEXT "Active Class and Subclass:"
FLOC 1 1
ENDTEXT
```

```
FIELD class
FLOC 5 4
ENUMSET 19
```

```
OPTIONS prettyprint
POP "class"
ENDFIELD

# option

FIELD subcolor
FLOC 2 7
COLOR 2 1
ENDFIELD

FIELD subclass
FLOC 5 7
ENUMSET 19
OPTIONS prettyprint ownerdrawn
POP "subclass"
ENDFIELD
```

Arguments

t_restrict	This optional argument is a string that indicates class and subclass restrictions if the form contains "class" and "subclass" popup fields that have not been overridden with calls to axlFormBuildPopup. Possible values are:
"NONE"	- no restrictions
"TEXT"	- only layers that allow text
"SHAPES"	- only layers that allow shapes
"RECTS"	- only layers that allow rectangles
"ETCH"	- only etch layers
"ETCH_PIN_VIA"	- only etch, pin, and via layers
"ETCH_NO_WIREBOND"	- only non-wirebond etch layers

See also [axlFormCreate](#) on page 502 for further details.

Value Returned

r_form Upon success, *r_form* is returned.

nil Failure due to one of the following:

No interactive command is active or the active command is not of the type AXL registered interactive.

AXL Forms code encounters an error.

Example

See swap component example:

```
<install_dir>/share pcb/etc/skill/examples/swap
```

axlDrawObject

```
axlDrawObject(  
    lo_dbid  
)  
⇒ t/nil
```

Description

Processes a list of *dbids*.

Redraws any objects that were erased by `axlEraseObject`.

Arguments

lo_dbid List of *dbids* or one *dbid*.

Value Returned

t One or more objects drawn.

nil No valid *dbids* or all objects already at desired display state.

axlDynamicsObject

```
axlDynamicsObject (
    lo_dbid
)
⇒ t/nil
```

Description

Adds list of objects to the cursor buffer. These objects are attached to the cursor in xor mode.

Note: Adding too many objects to the cursor buffer dramatically affects performance.

Arguments

<i>lo_dbid</i>	List of AXL <i>dbids</i> or single <i>dbid</i> .
----------------	--

Value Returned

t	One or more objects added to the cursor buffer.
---	---

nil	No objects added to the cursor buffer.
-----	--

axlEraseObject

```
axlEraseObject(  
    lo_dbid  
)  
⇒ t/nil
```

Description

Processes a list of *dbids* and erases them. Typically used with `axlDynamicsObject` to erase objects before attaching them to the cursor. Any objects erased are restored to their visibility when calling AXL shell or terminating the SKILL program.

Arguments

lo_dbid List of *dbids* or one *dbid*.

Value Returned

t One or more objects erased.

nil No valid *dbids* or all objects already at desired display state.

axlControlRaise

```
axlControlRaise(  
    g_option  
)  
⇒ t/nil
```

Description

Raises a tab in the control panel to the top. If you use this at the start of an interactive command, you override the environment variable, `control_auto_raise`.

Arguments

g_option Supported symbols are: 'options, 'find, 'visibility, and nil. nil returns a list of supported symbols.

Value Returned

t Tab raised to top in control panel.

nil Unknown symbol.

Example

```
axlControlRaise('options)
```

Raises the option panel to the top.

axlEnterEvent

```
axlEnterEvent(  
    l_eventMask  
    t_prompt  
    g_snap  
)  
⇒ r_eventId
```

Description

A lower level event manager than other `axlEnter` functions. Provides a Skill program with more user event details. See [Table 8-2](#) on page 362 for a list of events with descriptions.

Returns event structure containing the attributes described in [Table 8-1](#) on page 361. Event occurrence controls what attributes are set by all event types, and sets the `objType` and `time` attributes.

Table 8-1 Event Attributes

Attribute Name	Type	Description
<code>objType</code>	string	Type of object, in this case <code>event</code>
<code>type</code>	symbol	Event occurrence
<code>xy</code>	point	Location of mouse
<code>xySnap</code>	point	Location of mouse snapped to grid.
<code>command</code>	int/symbol	Returns the callback item of <code>axlUIPopUpDefine</code>
<code>time</code>	float	time stamp (seconds.milliseconds)

Note: Do not put a default handler in your case statement since the event model will change in future releases.

Table 8-2 Events

Event	Description	Attributes/Mask
PICK	User has selected a point (equal to axlEnterPoint)	
PICK_EXTEND	Same as PICK except has extend keyboard modifier.	
PICK_TOGGLE	Same as PICK except has toggle keyboard modifier.	xy, xySnap
DBLPICK	User has double picked at a location.	
DBLPICK_EXTEND	Same as DBLPICK except has extend keyboard modifier.	
DBLPICK_TOGGLE	Same as DBLPICK except has toggle keyboard modifier.	xy, xySnap
STARTDRAG	User starts a drag operation.	
STARTDRAG_EXTEND	Same as STARTDRAG except has extend keyboard modifier.	
STARTDRAG_TOGGLE	Same as STARTDRAG except has toggle keyboard modifier.	xySnap
STOPDRAG	User terminated the drag operation.	
STOPDRAG_EXTEND	Same as STOPDRAG except has extend keyboard modifier.	
STOPDRAG_TOGGLE	Same as STOPDRAG except has toggle keyboard modifier.	xy, xySnap, command

Table 8-2 Events

Event	Description	Attributes/Mask
DONE	User requests the command to complete.	This event cannot be masked.
CANCEL	Respond to this event by terminating your Skill program (don't call any more <i>axlEnter</i> functions.)	This event cannot be masked.

Notes

- You will get `PICK` before `DBLPICK` events. To differentiate between a `PICK` and `DBLPICK`, highlight the selected object. Do not output informational messages or perform time consuming processing.
- Never prompt user for a double click. Instead, format prompts for the next expected event.
- Events dispatched from `axlEnterEvent` are scripted by the system if scripts are enabled.
- The `done` and `cancel` callbacks optionally defined in `axlCmdRegister` are called before the `DONE` and `CANCEL` events are returned.
- The `extend` keyboard modifier is obtained by holding the `Shift` key while performing the mouse operation.
- The `toggle` keyboard modifier is obtained by holding the `Control` key while performing the mouse operation.



Tip

You can program more easily by providing a single mask set for your command and by not attempting to change the mask set after each event.

Arguments

<i>l_eventMask/nil</i>	List of events to expect.
<i>t_prompt/nil</i>	User prompt. If <i>nil</i> , the default prompt is used.
<i>g_snapGrid</i>	If <i>t</i> , grid snapping is enabled while the function is active. Otherwise no grid snapping is allowed. This affects the <i>xySnap</i> value that is returned as well as dynamics and the <i>xy</i> readout. If <i>nil</i> , <i>xySnap</i> is not snapped to the grid and is the same as <i>xy</i> .

Value Returned

<i>r_eventId</i>	Event structure containing attributes.
------------------	--

Example

```
let( (eventMask event, loop)
    eventMask = list( 'PICK 'DBLPICK )
    loop = t
    while( loop
        event = axlEnterEvent(eventMask, nil)
        case(event->type

            ( 'PICK
                ...
            )
            ( 'DBLPICK
                ...
            )
            ( 'DONE
                ; cleanup
```

axlEventSetStartPopup

```
axlEventSetStartPopup(  
    [s_callback]  
)  
⇒ t/nil
```

Description

Sets a SKILL callback function called prior to a popup being displayed on the screen. Allows AXL applications to reset the popup (see `axlUIPOPUPSetsee`), thus providing context sensitive popups support.

The callback function is passed a list structure the same as the return list in `axlEnterEvent`. Use this function with `axlEnterEvent`.

The callback function is removed when an AXL application is finished. Set this at the application start, if needed.

Arguments

<code>s_callback</code>	AXL callback function.
<code>none</code>	Unsets the callback function which disables the callback mechanism.

Value Returned

<code>t</code>	Set SKILL callback function.
<code>nil</code>	Failed to set SKILL callback function.

Example

```
(defun startpopupcallback (event)
  ...
  newpopup = get a new popup based on event x,y values
  axlUIPopupSet(newpopup)
)
axlEventSetStartPopup('startpopupcallback')
...

let( (eventMask event, loop)
  eventMask = list( 'PICK 'DBLPICK )
  loop = t
  while( loop
    event = axlEnterEvent(eventMask, nil)
    case(event->type

      ('PICK
       ...)
      ('DBLPICK
       ...)
      ('DONE
       loop = nil)
      ('CANCEL
       loop = nil)
    )
  )
)

...
axlEventSetStartPopup()
```

Typically used in conjunction with `axlEnterEvent`.

axlGetTrapBox

```
axlGetTrapBox(  
    l_point  
)  
⇒ l_window/nil
```

Description

Returns coordinates of the *Find* window.

Arguments

l_point Listing of the *x* and *y* coordinates

Value Returned

l_window ((x_l y_l) (x_u y_u)) - List of corner coordinates of the *Find* window.
 (x_l y_l) - List containing *x* and *y* coordinates of the lower left corner.
 (x_u y_u) - List of the *x* and *y* coordinates of the upper right corner.

nil *l_point* is null or in an incorrect format.

axlRatsnestBlank

```
axlRatsnestBlank(  
    rd_net  
)  
⇒ t/nil
```

Description

Blanks all ratsnest lines in a net.

Arguments

rd_net *dbid* of a net

Value Returned

t Ratsnest lines are blanked.

nil Ratsnest lines are not blanked.

axlRatsnestDisplay

```
axlRatsnestDisplay(  
    rd_net  
)  
⇒ t/nil
```

Description

Displays all ratsnest lines in a net.

Arguments

rd_net *dbid* of a net

Value Returned

t Ratsnest lines are displayed.

nil Ratsnest lines are not displayed.

axlShowObjectToFile

```
axlShowObjectToFile(  
    lo_dbid  
    [t_file_name]  
)  
⇒ (t_file_name x_width x_line_count)
```

Description

Creates a temporary file with show element information on *dbids* specified in *lo_dbid*.

Arguments

lo_dbid List of *dbids* or a single *dbid*.

t_file_name File name to use instead of a temporary file.

Value Returned

List of items describing the file created (*t_file_name* *x_width* *x_line_count*):

t_file_name Name of the temporary file.

x_width Width, in characters, of the widest text line.

x_line_count Number of lines in the file.

nil Could not create file.

axlUICmdPopupSet

```
axlUICmdPopupSet(  
    r_popup  
)  
⇒ r_prevPopup
```

Description

Sets up a popup menu with all menu items required throughout the execution of the command. Call during the command's initialization process. Use of this procedure modifies the behavior of axlUIPopUpSet so that it makes unavailable all popup items not in the defined popup.

Adds a `cmdPopupId` property to AXL user data which restores popup entries whenever the AXL command state is restored. The command popup is cleared when the Skill command ends.

Arguments

<i>r_popup</i>	Popup handle, obtained by calling <code>axlUIPopUpDefine</code> . A <code>nil</code> value turns off this popup.
----------------	--

Value Returned

<i>r_prevPopup</i>	Popup set previously defined.
--------------------	-------------------------------

Note: This procedure does the same as `axlCmdPopupSet` for non-WXL UI's.

axlZoomToDbid

```
axlZoomToDbid(  
    o_dbid/lo_dbid  
    g_always  
)  
⇒ t/nil
```

Description

Processes a list of *dbids* and centers and zooms the display around them. Zoom is done so objects extents fill about 20% of the display.

Note: You should highlight the objects.

Note: If more than 20 objects are passed no zoom is done.

Arguments

<i>o_dbid</i>	List of <i>dbids</i> or one <i>dbid</i> .
<i>g_always</i>	If <i>t</i> , then ignores NO_ZOOM_TO_OBJECT environment variable.

Value Returned

<i>t</i>	One or more objects zoomed.
<i>nil</i>	No valid <i>dbids</i> or all objects are already at desired display state.

axlMakeDynamicsPath

```
axlMakeDynamicsPath(  
    lo_dbidSegs  
)  
⇒ r_path/nil
```

Description

Takes a list of *dbids* segments and turns them into an *r_path*, which normally would require using `axlDBCCreate` and `axlPoly` functions.



Passing other o_dbid types causes unpredictable results.



A circle is an arc segment with the same end points.

Arguments

<i>lo_dbidSegs</i>	List of <i>dbids</i> of line or arc segments of type "line" or "arc"
--------------------	--

Value Returned

<i>r_path</i>	<i>dbid</i> of <i>r_path</i> constructed of the segments.
---------------	---

<i>nil</i>	No <i>r_path</i> constructed due to incorrect arguments.
------------	--

Example

Simple *r_path* segment with a width of 20.

```
a = axlMakeDynamicsPath('((10:10 100:100 15.1)))
```

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor Interface Functions

PCB Editor Command Shell Functions

Overview

This chapter describes the AXL-SKILL functions that access the PCB Editor environment and command shell.

Command Shell Functions

This section lists PCB Editor command shell functions.

axlGetVariable

```
axlGetVariable(  
    t_variable  
)  
⇒ t_value/nil
```

Description

Requests the value of the specified PCB Editor environment variable, *t_variable*. Returns a list containing the string assigned to the variable or *nil* if the variable is currently not set in PCB Editor. Use `axlGetVariableList` where the variable stores a list of items (such as a PATH variable) to preserve any spaces in each item.

Note: Variable names are case insensitive.



Variable names and values can change from release to release.

Arguments

t_variable String giving name of the PCB Editor environment variable.

Value Returned

t_value List containing string value of the PCB Editor environment variable.

nil Variable not set.

Example 1

```
menu = axlGetVariable("menuload")  
      ==> "geometry"  
psmpath = axlGetVariable("psmpath")  
      ==> ". symbols"
```

- Gets value of the current menu loaded.

Variable name is *menuload*.

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor Command Shell Functions

- Gets the value of the library search path *libpath*.

axlGetVariableList

```
axlGetVariableList(  
    t_variable/nil  
)  
==> t_value/lt_value/nil
```

Description

Requests the value of the specified PCB Editor environment variable, *t_variable*. Unlike `axlGetVariable` this returns a list of strings, if the variable is an array, such as one of PCB Editor's path variables. If variable is a single item, the return is the same as `axlGetVariable`.

Since path variables can contain spaces, using the `axlGetVariable` interface and then Skill's `parseString` to break them back to the component pieces will not give the correct result.

Note: Variable names are case insensitive.

Note: Variable names and values can change from release to release.

Arguments

t_variable Name of the PCB Editor environment variable.

nil If *nil*, returns a list of all set variables in PCB Editor.

Value Returned

t_value String value of the PCB Editor environment variable.

nil Returns *nil* if the variable is not set.

lt_names If passed, *nil* returns list of variable names

See also `axlGetVariable`.

Example

Gets the value of the Package Symbol search path:

```
path = axlGetVariable( "psmpath" )  
==> ( "." "symbols" "/cds/root/share pcb/allegrolib/symbols" )
```

Allegro PCB and Package User Guide: SKILL Reference
PCB Editor Command Shell Functions

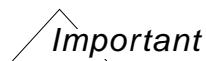
axlSetVariable

```
axlSetVariable(  
    t_variable / lt_variable  
    [g_value]  
)  
⇒ t/nil
```

Description

Sets the PCB Editor environment variable with name given by the string *t_variable* to the value *g_value*. *g_value* can be a string, int, t, or nil. Returns the string assigned to the variable or nil if the variable cannot be set in PCB Editor.

Note: 511 is the maximum list long (*lt_variable*).



Variable names and values can change from release to release.

Arguments

t_variable String giving the name of the PCB Editor environment variable.

g_value Value to which the environment variable is to be set. Can be a string, int, t, or nil.

Value Returned

t Environment variable set.

nil Environment variable not set.

Example

Sets new library search path libpath.

```
(axlSetVariable "libpath" "/mytools/library")  
⇒ t  
libraryPath = (axlGetVariable "libpath")  
⇒ "/mytools/library"
```

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor Command Shell Functions

Using list mode.

```
axlSetVariable( "psmpath"  '( ."  "symbols"))
    ==> t
axlGetVariableList( "psmpath")
    ==> ( ."  "symbols)
```

axlUnsetVariable

```
axlUnsetVariable(  
    t_variable  
)  
⇒ t
```

Description

Unsets the PCB Editor environment variable with the name given by the string *t_variable*. The value of the named variable becomes *nil*.



Variable names and values can change from release to release.

Arguments

t_variable String giving the name of the PCB Editor environment variable.

Value Returned

t Always returns *t*.

Example

```
(axlUnsetVariable "libpath")  
⇒ "/mytools/library"  
libraryPath = (axlGetVariable "libpath")  
⇒ nil
```

Clears the library path *libpath* when its current value is */mytools/library*.

axlShell

```
axlShell(  
    t_command  
)  
⇒ t
```

Description

Issues the PCB Editor command string *t_commands* to the connected editor. You can chain commands. This call is synchronous.



This function might not be portable across PCB Editor releases.

Arguments

t_command PCB Editor shell command or commands.

Value Returned

t Always returns *t*.

Example

```
(axlShell "status")  
⇒ t
```

Displays PCB Editor Status form from AXL-SKILL.

axlGetAlias

```
axlGetAlias(  
    t_alias/nil  
)  
⇒ t_value/lt_names/nil
```

Description

Requests the value of the specified PCB Editor alias, *t_alias*. If given a *nil*, returns a list of aliases currently set.

Arguments

t_alias Name of the PCB Editor environment alias.

Value Returned

t_value String value of the PCB Editor environment alias.

lt_names List of alias names.

nil Alias not set.

Example 1

```
alias = axlGetAlias("SF2")  
⇒ "grid"
```

Gets the value of the alias assigned to shifted function key F2.

Example 2

```
list_alias = axlGetAlias(nil)  
⇒ ("F2" "F3" "F4" ...)
```

Returns all set aliases.

axlIsProtectAlias

```
axlIsProtectAlias(  
    t_alias  
)  
⇒ t/nil
```

Description

Tests if the alias is marked protected.

Arguments

<i>t_alias</i>	Name of the PCB Editor environment alias.
----------------	---

Value Returned

<i>t</i>	Alias is protected.
----------	---------------------

<i>nil</i>	Alias is unprotected or not set.
------------	----------------------------------

axlProtectAlias

```
axlProtectAlias(  
    t_alias  
    t/nil  
)  
⇒ t/nil
```

Description

Controls the read-only attribute of an alias.

Notes

- Do not unprotect F1 as this is fixed to *Help* by the operating system.
- You must define the alias before you can protect it.

Arguments

t_alias Name of the PCB Editor environment alias.

t/nil *t* protects the alias, and *nil* unprotects the alias.

Value Returned

t Successfully protected or unprotected the alias.

nil Alias is not set, or the function received invalid data.

Example

```
axlProtectAlias( "F2" t)
```

Protects the F2 function key.

axlReadOnlyVariable

```
axlReadOnlyVariable(  
    t_variable  
    [g_Enable]  
)  
==> t/nil
```

Description

This sets, unsets or queries the read-only state of a PCB Editor environment variable. When you set a variable as read-only, it cannot be changed.

Note: Variable names are case insensitive.

Arguments

t_variable The name of the PCB Editor environment variable.

g_Enable *t* to set read-only; *nil* to make writable and do not provide if using to test variable read-only state.

Value Returned

t/nil In query mode (*no g_Enable option*) returns *t* if variable is read-only and *nil* if not. If changing the read-only mode, returns *t* if successful and *nil* if variable is not currently set.

See also `axlGetVariable`.

Examples

The following example:

- Sets `psmpath` to read-only.
- Queries the setting.
- Resets `psmpath` to writable.

Query the setting:

```
axlReadOnlyVariable("psmpath" t)
```

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor Command Shell Functions

```
axlReadOnlyVariable( "psmpath" )
=> t
axlReadOnlyVariable( "psmpath" nil)
axlReadOnlyVariable( "psmpath" )
=> nil
```

Query all read-only variables:

```
axlReadOnlyVariable( "fxf" t)
axlReadOnlyVariable( "psmpath" t)
axlReadOnlyVariable(nil)
=> ( "psmpath" "fxf" )
```

axlSetAlias

```
axlSetAlias(  
    t_alias  
    g_value  
)  
⇒ t/nil
```

Description

You can set the PCB Editor environment alias with the name given by the string *t_alias* to the value *g_value* using the `axlSetAlias` function. *g_value* can be a string, int, t, or nil. Returns the string assigned to the alias or nil if the alias cannot be set in PCB Editor.

You can use function keys F2-F12, most Alpha-numeric keys with the control modifier (although Control-C V and X are reserved for copy, paste, and cut) and the Navigation Keys (Home, Up arrow, Esc, etc.) You can modify these items as shown:

Modifier	Indicator	Example
Shift	S	SF2
Control	C (function keys)	CF2
Control	~ (alpha-numeric)	-N
Meta	A	AF2

Modifiers may be combined as shown in these examples:

CSF2	Control-Shift F2
ASF2	Meta-Shift F2
CAF2	Control-Meta F2
CASF2	Control-Meta-Shift F2
~SZ	Control-Shift Z
SUp	Shift-Up Arrow
CUp	Control-Up Arrow

Notes:

- Alias settings only apply to the current session. They are not saved to the user's local env file.
- Alias changes do not affect programs launched from PCB Editor, for example, import logic, refresh_symbol.

Arguments

t_alias Name of the PCB Editor environment alias.

g_value Value to which the environment alias is to be set. Can be a string or nil.

Value Returned

t Alias set.

nil Invalid data or alias is marked read-only.

Example 1

```
axlSetAlias( "F2" "save" )
```

Sets the F2 function key to the save command.

Example 2

```
axlSetAlias( "~S" nil )
```

Unsets the Ctrl-S alias.

User Interface Functions

Overview

This chapter describes the AXL/SKILL functions you use to confirm intent for an action, prompt for text input, display ASCII text files, and flush pending changes in the display buffer.

Window Placement

PCB Editor encourages you to place windows in an abstract manner. For example, when you open a form, instead of specifying (x,y) coordinates you give a list of placement options. PCB Editor then calculates the placement location. An advantage of this method is that all windows automatically position themselves relative to the main PCB Editor window. Windows always position entirely onscreen even in violation of your placement parameters.

The following form placement options (strings with accepted abbreviations in parentheses) are available:

north(n)	northeast(ne)	east(e)	southeast(se)
south(s)	southwest(sw)	west(w)	northwest(nw)
center(c)			

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

In addition you can modify the placement options with the following parameters:

Inner or Outer	Places the placement rectangle to the outside or the inside of the main window. The default is <code>inner</code> .
Canvas or Window	Uses the canvas (drawing) area or the entire window for the placement rectangle. The default is <code>Window</code> .
Border or NoBorder (Default Border)	Leaves a slight border around the placed window. If <code>noborder</code> is set, the window is set directly against the placement rectangle. The default is <code>Border</code> .
MsgLines (Default 1)	Sets the number of message lines at bottom of the placed window to 0 or 1.

Note: Only `forms` supports this parameter.

Syntax:

```
msglines #
```

Using Menu Files

You can use drawing menus, symbol menus, and shape menus in PCB Editor. You can find the default PCB Editor menu files in:

```
<cdsroot>/share pcb/text/cuimenus
```

Note: Menu names and contents change from release to release.

Depending on the tools and software version you have installed, you have some or all of the menu files shown:

Table 10-1 PCB Editor Menu Files

File Name	Description
aplayout.men	Allegro Package Designer 610 drawing
apdsymbol.men	Allegro Package Designer 610 symbol
cbdsymbol.men	Symbol menu for all PCB Editor products
allegro.men	PCB Editor menu for all PCB Editor products
lbrlayout.men	Allegro PCB Librarian 610 drawing
padlayout.men	Pad Designer in the board graphics editor
padlaystn.men	Pad Designer (standalone)
pqlayout.men	Allegro PCB SI 610
viewlayout.men	Allegro Physical Viewer 610
sqpkq.men	Allegro Package SI 610
specctraquest.men	Allegro SI
sqpkg3d.men	Allegro Package SI 620
xlibsymbol.men	Allegro PCB Librarian 610 Symbol Editor

Dynamically Loading Menus

All tools support overriding their default menus by putting your menu file before the default Cadence menu file via the MENUPATH. Programs that support AXL-Skill allow menus to be dynamically changed while the program is running. You do this using the axlUIMenuLoad Skill function. This is not supported in *allegro_pc*b and *allegro_viewer*.

Understanding the Menu File Format

You can have only one menu definition per file. The following shows the menu syntax in BNF format. The use of indentation reflects hierarchy in the .men file.

Menu file grammar reflects the following conventions:

Convention	Description
[]	Optional
{ }	May repeat one or more times.
< >	Supplied by the user
	Choose one or the other
:	Definition of a token
CAPS	Items in caps are keywords

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

The following defines the menu file format:

FILE:

```
[comment]
[ifdef]
<name>MENU DISCARDABLE
BEGIN
    {popup}
END

popup:
POPUP "<display>"
BEGIN
    {MENUITEM "<display>"           "<command>"}
    [{separator}]
    {[popup]}
END

{[//]}      - comment lines

separator:
MENUITEM SEPARATOR
- this inserts a separator line at this spot in the menu. This is not
supported at the top level menubar.

name:      This text is ignored. Use the file name without the extension.
comment:   Double slash (//) can be used to start a comment.
display:   Text shown to the user.
          & -This is used to enable keyboard access to the menus. For this to
          work, each menu level must have a unique key assigned to it. Use
          double ampersand (&&) to display a "&".
          ... -The three dots convention signifies that this command displays a
          form.

command:   This is any Allegro command, sequence of Allegro commands, or Skill
statement. The Allegro command parser acts on this statement so it offers
considerable flexibility.

ifdef:     Use #ifdef/#endif and #ifndef/#endif to make items conditionally appear
in the menu, depending on whether or not a specified environment variable
is set.

A #ifdef causes the menu item(s) to be ignored unless the environment
variable is set. A #ifndef causes the menu item(s) to be ignored if the
environment variable is not set. You must have one #endif for each #ifdef
or #ifndef to end the block of conditional menu items. Also, #ifdef,
#ifndef, and #endif must start at the first column of the line in the
menu file.
```

The syntax for #ifdef follows:

```
#ifdef <env variable name>
[menu items which appear if the env variable is set]
#endif

#ifndef <env variable name>
[menu items which appear if the env variable is not set]
#endif
```

The items between the #if[n]def/#endif can be one or more MENUITEMS or could be a POPUP.

Example 1

```
#ifdef menu_enable_export
    POPUP "&Export"
    BEGIN
        MENUITEM "&Logic...", "feedback"
    END
#endif
```

The *Export* popup appears in the menu only if the `menu_enable_export` environment variable is set.

Example 2

```
#ifndef menu_disable_product_notes
    MENUITEM "&Product Notes", "help -file algpn"
#endif
```

The *Product Notes* menu item appears in the menu only if the `menu_disable_product_notes` environment variable is NOT set.

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

Example 3 - Simple Menu Example

```
DISPLAY (indents reflect the various pulldown levels)
        File           Help
            Open          Contents
            Export         Product Notes
                Logic       Known Problems and Solutions
            Exit
-----
            About Allegro...

FILE:
simple MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Open" ,                      "open"
        POPUP "&Export"
        BEGIN
            MENUITEM "&Logic..." ,             "feedback"
            END
            MENUITEM "&Exit" ,                  "exit"
        END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Contents" ,                 "help"
        MENUITEM "&Product Notes" ,            "help -file algpn"
        MENUITEM "&Known Problems and Solutions" , "help -file alkpns"
        MENUITEM SEPARATOR
        MENUITEM "&About Allegro..." ,        "about"
    END
END
```

A simple menu and the simple file required to display the menu.

AXL-SKILL User Interface Functions

This section lists the user interface functions.

axlCancelOff

See axlCancelOn.

axlCancelOn

```
axlCancelOn(  
)  
⇒ t  
axlCancelOff(  
)  
⇒ t  
axlCancelTest(  
)  
⇒ t/nil
```

Description

Allows Skill code to test for when a user clicks *Cancel*.

When cancel is enabled, the traffic light is yellow.

Although you can nest cancel calls, you should make an equal number of cancel off calls as cancel on calls.

Note: To avoid problems, always place the cancel on/off call pairs in the same function.

These calls do not work from the Skill or PCB Editor command line because PCB Editor immediately disables cancel when exiting the Skill environment to prevent the system from hanging.

Notes:

- Only enable cancel processing when you are sure there is no user interaction. Having cancel enabled when the user has to enter information is not supported and will hang the system.
- Calling `axlCancelTest` can adversely impact your program's performance.

Arguments

None

Value Returned

Only axlCancelTest returns meaningful data.

t	User click cancel.
nil	User did not click cancel.

Examples

```
count = 0
axlCancelOn()
while ( count < 50000 && !axlCancelTest()
    printf("Count = %d\n" count)
    count++
)
axlCancelOff()
```

axlCancelTest

See `axlCancelOn`.

axlMeterCreate

```
axlMeterCreate(  
    t_title  
    t_infoString  
    g_enableCancel  
    [t_formname]  
    [t_infoString2]  
    [g_formCallback]  
-> t/nil
```

Description

Starts progress meter with optional cancel featur

Note: Always call `axlMeterDestroy` when done with meter.

Arguments

<i>t_title</i>	Title bar of meter.
<i>t_infoString</i>	One line of 28 characters used for anything you want (can be updated at meter update).
<i>g_enableCancel</i>	<i>t</i> enable the application <i>Stop</i> button on graphical UI-based applications. When enabled and the user picks the <i>Stop</i> button, a true is returned by the call to <code>axlMeterIsCancelled()</code> .
<i>t_formname</i>	(Optional) The name of an alternate form that can be used with these functions which has an info field named <i>progressText</i> and a progress field named <i>bar</i> . <code>axlMeterIsCancelled</code> will also notice if a <i>Cancel</i> menu button has been pressed. If you do not give a form name <code>axlprogress.form</code> will be used.
<i>t_infoString2</i>	(Optional) By Default "".
<i>g_formCallback</i>	(Optional) The name of a Callback function that you want called for any buttons or fillins etc you may have on your form. This works the same as <i>g_formAction</i> in <code>axlFormCreate</code> .

Value Returned

t On success; otherwise nil.

See also

`axlMeterCreate`, `axlMeterIsCancelled`, `axlMeterDestroy` and `axlFormCreate`

Example

```
axlMeterCreate("SigNoise Design Audit", "", t)
total = <total nets>
done = 0
while(<still next net> && (!axlMeterIsCancelled()))
    < do work >
    axlMeterUpdate( (100 * ++done)/total
        sprintf(nil "Check %d of %d nets" done total))
)
axlMeterDestroy()
```

axlMeterDestroy

```
axlMeterDestroy ( ) -> t/nil
```

Description

Closes the progress meter form and shuts off Cancel mode if enabled.

Arguments

None

Value Returned

t If meter was destroyed; otherwise nil.

See also

[axlMeterCreate](#)

axlMeterUpdate

```
axlMeterUpdate(  
    x_percentDone  
    t_infoString  
    [t_infoStr2]  
) -> t/nil
```

Description

Updates progress meter bar and/or info text. The percent done and/or the info string may be updated.

Arguments

<i>x_percentDone</i>	Integer task percent done (0-100)
<i>t_infoString</i>	Update text for progress meter info text line. Value is one of: nil - leave info text as it is. "" - clear info string field.
<i>newText</i>	Update field with new text.
<i>t_infoStr2</i>	(optional) Text for second line.

Value Returned

<i>t</i>	On success; otherwise nil.
----------	----------------------------

See also

[axlMeterCreate](#)

axlMeterIsCancelled

```
axlMeterIsCancelled(  
    ) -> t/nil
```

Description

If cancel was enabled at meter creation, the status of cancel is returned (*t* if cancelled; otherwise *nil*).

If a field named *Cancel* was hit, it is cancelled

Arguments

None

Value Returned

t If meter was cancelled; otherwise *nil*.

See also

[axlMeterCreate](#)

axlUIMenuLoad

```
axlUIMenuLoad (
    t_menufile
)⇒ t_previousMenuName/nil
```

Description

Loads the main window menu from the file *t_menuFile*. Adds a default menu file name extension if *t_menuFile* has none. The `MENUPATH` environment variable is used to locate the file if *t_menuFile* does not include the entire path from the root drive.

Note: This function is for the Windows-based GUI only.

Arguments

<i>t_menuFile</i>	Name of the file to which the menu is dumped. If <i>t_menuFile</i> is <code>nil</code> , the file name is based on the program's default menu name, which may vary based on the current state of the program.
-------------------	---

Value Returned

<i>t_previousMenuName</i>	Name of the previous menu.
---------------------------	----------------------------

<code>nil</code>	Menu not be located.
------------------	----------------------

axlUIPopupMenu

```
axlUIPopupMenu (   
    t_MenuFile  
 ) => t_previousMenuName / nil
```

Description

Dumps the main window's current menu to the file *t_menuFile*. Adds default menu file name extension if *t_menuFile* has none.

Notes:

- There is no user interaction when an existing file is overwritten.
- This function is for the Windows-based GUI only.

Arguments

<i>t_menuFile</i>	Name of the file to which the menu is dumped. If <i>t_menuFile</i> is <i>nil</i> , the file name is based on the program's default menu name, which may vary based on the current state of the program.
-------------------	---

Value Returned

<i>t_previousMenuName</i>	Full name of the file that is written.
---------------------------	--

nil	No file is written.
-----	---------------------

axlUIColorDialog

```
axlUIColorDialog(  
    r_window/nil  
    l_rgb  
) -> l_rgb/nil
```

Description

Invokes standard color selection dialog box. You must provide a parent window, PCB Editor defaults to the main window of the application.

The *l_rgb* is a red, green, or blue palette list. Each item is an integer between the values of 0 and 255. 0 indicates color is off, and a value of 255 indicates color is completely on. For example, 255 255 255 indicates white.

Arguments

<i>r_window</i>	Parent window. If <i>nil</i> , use main program window. Return handle of <code>axlFormCreate</code> is of type <i>r_window</i> .
<i>l_rgb</i>	Seeded red, green, or blue.

Value Returned

<i>l_rgb</i>	User selected values.
<i>nil</i>	User canceled dialog box.

See also `axlColorSet`, `axlColorGet`.

Examples

Get color 1 and change it:

```
rgb = axlColorGet(1)  
rgb = axlUIColorDialog(nil rgb)  
when(rgb  
    axlColorSet(1 rgb)  
    axlVisibleUpdate(t))
```

axlUIConfirm

```
axlUIConfirm(  
    t_message  
)  
⇒ t
```

Description

Displays the string *t_message* in a dialog box.

The user must respond before any further interaction with PCB Editor. Useful mainly for informing the user about a severe fatal error before exiting your program. Use this blocker function very rarely.

Note: If environment variable `noconfirm` is set, we immediately return.

Arguments

<i>t_message</i>	Message string.
------------------	-----------------

Value Returned

<i>t</i>	Always returns <i>t</i> .
----------	---------------------------

Example

```
axlUIConfirm( "Returning to Allegro. Please confirm." )
```

Informs the user when a significant transition is being made.

A dialog box is displayed. The user must select *OK* before control returns to the program.

axlUIControl

```
axlUIControl(  
    s_name  
    [g_value]  
)  
==> g_currentValue/ls_names
```

Description

Inquires and sets the value of graphics. If setting a value, the return is the old value of the control. If an active form displaying the current settings is open, it may not update. Settings currently supported:

```
{UIScreen, NULL, "screen" },  
{UIVScreen, NULL, "vscreen"},  
{UIMonitors, NULL, "monitors"},  
{UIPixel2DB, NULL, "pixel2UserUnits"},  
Name: screen  
Value: (x_width x_height)  
Set?: No
```

Description: Retrieves the screen's width and height in pixels

Equiv: none

Side Effects: none

```
Name: vscreen  
Value: (x_width x_height)  
Set?: No
```

Description: Retrieves the screen's virtual width and height in pixels. This will not be the same as 'screen' if running Windows XP and enabled monitor spanning option. Also requires multiple monitors and graphic card(s) capable of supporting multiple monitors.

Equiv: none

Side Effects: On UNIX always returns the same size as screen.

```
Name: monitors  
Value: x_number  
Set?: No
```

Description: Retrieves the number of monitors available.

Equiv: none

Side Effects: On UNIX always returns 1 since we currently do not support multi-monitors on UNIX.

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

Name: pixel2UserUnits

Value: f_number

Set?: No

Description: Returns number user units per pixel taking into account the current canvas size and zoom factor. Changes with the current zoom factor.

Equiv: none

Side Effects: none

Arguments

s_name Symbol name of control. nil returns all possible names.

s_value Optional symbol value to set. Usually a t or a nil.

Value Returned

ls_names If name is nil then returns a list of all controls.

Examples:

Get screen size:

```
size = axlUIControl('screen)
-> (1280 1024)
```

Get pixel to user units:

```
axlUIControl('pixel2UserUnits)
-> 17.2
```

axlUIPrompt

```
axlUIPrompt(  
    t_message  
    [t_default]  
)  
⇒ t_response/nil
```

Description

Displays the string *t_message* in a form. The user must type a response into the field. Displays the argument *t_default* in brackets to the left of the field. The user presses the *Return* key or clicks the *OK* button in the window to accept the value of *t_default* as the function return value. If the user selects the *Cancel* button, the function returns *nil*.

This function is a blocker. The user must respond before any further interaction with PCB Editor.

Arguments

<i>t_message</i>	Message string displayed.
<i>t_default</i>	Default value displayed to the user and returned if user presses only the <i>Return</i> key or clicks <i>OK</i> .

Value Returned

<i>t_response</i>	User response or default value.
<i>nil</i>	User selected <i>Cancel</i> .

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

Example

```
axlUIPrompt( "Enter module name" "demo" )  
⇒ "mymcm"
```

Prompts for a module name with a default demo. Typing mymcm overrides the default.

axlIsViewFileType

```
axlIsViewFileType(  
    g_userType  
)  
⇒ t/nil
```

Description

Tests whether *g_userType* is a long message window type.

Arguments

g_userType Argument to test.

Value Returned

t *g_userType* is of type *r_windowMsg*.

nil *g_userType* is not of type *r_windowMsg*.

Example

```
logWindow =  
    axlUIViewFileCreate( "batch_drc.log" "Batch DRC Log" t)  
    axlIsViewFileType(logWindow)  
⇒ t
```

- Creates a window using `axlUIViewFileCreate` (See [axlUIViewFileCreate](#) on page 416.)
- Tests whether the window is a view file type.
- Returns t.

axlUIViewFileCreate

```
axlUIViewFileCreate(  
    t_file  
    t_title  
    g_deletefile  
    [lx_size]  
    [lt_placement]  
)  
⇒ r_windowMsg/nil
```

Description

Opens a file view window and displays the ASCII file *t_file* using the arguments described below.

Arguments

<i>t_file</i>	Name of the ASCII file to display.
<i>t_title</i>	Title to be display in window title bar.
<i>g_deletefile</i>	Deletes the file when the user quits the window or another task reuses the window. Use this to delete files you want to discard, since this allows correct operation of the <i>Save</i> and <i>Print</i> buttons in the file view window.
<i>lx_size</i>	Initial size of the window in character rows and columns. The default is 24 by 80. Setting a large window size may cause unpredictable results.
<i>lt_placement</i>	Location of the window with respect to the PCB Editor window. The default location is centered on PCB Editor window.

Value Returned

r_windowMsg Window *r_windowMsg*.

nil *r_windowMsg* not displayed.

Example

```
logWindow =
    axlUIViewFileCreate( "batch_drc.log" "Batch DRC Log" t )
⇒ windowMsg:2345643
```

- Displays the batch DRC log file, saving the window id.
- Deletes the file `drc.log` when the user exits the window.
- The log file displays in a window.
- When the user chooses *Close*, deletes the file `batch_drc.log`.

axlUIViewFileReuse

```
axlUIViewFileReuse(  
    r_windowMsg  
    t_file  
    t_title  
    g_deleteFile  
)  
⇒ t/nil
```

Description

Reuses the existing view window *r_windowMsg* created by an earlier call to the function axlUIViewFileCreate, gives it the title *t_title*, and displays the ASCII file *t_file* in the window. If the user has already quit the window since its previous use, the window reopens at its old size and position.

Arguments

<i>r_windowMsg</i>	<i>dbid</i> of the existing view window created earlier with axlUIViewFileCreate.
<i>t_file</i>	Name of the ASCII file to display.
<i>t_title</i>	Title to display in window title bar.
<i>g_deleteFile</i>	Deletes file when the user quits the window or another task reuses the window. Use to delete files you want to discard, since this allows correct operation of the <i>Save</i> and <i>Print</i> buttons in the file view window.

Value Returned

<i>t</i>	File displayed.
<i>nil</i>	File not displayed.

Example

```
(axlUIViewFileReuse logWindow "ncdrill.log" "NC Drill Log" t)
```

- Displays the file `ncdrill.log`, reusing the window `logWindow` created when displaying `router.log` in the `axlUIViewFileCreate` example.
- Exiting the window deletes the file `ncdrill.log`.

axlUIYesNo

```
axlUIYesNo(  
    t_message  
)  
⇒ t/nil
```

Description

Provides a dialog box displaying the message *t_message*. Returns *t* if you choose Yes and *nil* for No.

This function is a blocker. The user must respond before any further interaction with PCB Editor.

Note:

- The dialog box also displays a *Help* button. Selecting this button returns *nil*, and has no other effect for this release of AXL-SKILL.
- If environment variable `noconfirm` is set, we immediately return *t*.

Arguments

t_message Message string to display.

Value Returned

t User responded Yes.

nil User responded No.

See also `axlUIConfirm`.

Examples

The following examples are a typical overwrite question.

```
axlUIYesNo( "Overwrite module?" )  
  
axlUIYesNo( "Overwrite module?" nil 'no )
```

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

```
axlUIYesNo( "Overwrite module?" "My Skill Program" )
```

A confirmer window is displayed. If the user selects Yes, the function returns t, otherwise it returns nil.

```
**/
```

```
list
axlUIYesNo(int argc, list *argv)
{
    char *str, *title;
    int dflt;

    str = axluGetString(NULL, argv[0]);
    title = (argc>1) ? axluGetString(NULL, argv[1]) : NULL;
    dflt = (argc>2) ? DfltResponse(argv[2]) : MN_YES;

    return(MNYesNoWTITLE(str, title, dflt) ? ilcT : ilcNil);
}

/*
#endif DOC_C
```

axlUIWExpose

```
axlUIWExpose(  
    r_window/nil  
)  
⇒ t/nil
```

Description

Opens and redispers a hidden or iconified window, bringing it to the front of all other current windows on the display. If `nil`, the main window is displayed.

Arguments

`r_window` Window *dbid*.

Value Returned

`t` Window opened and brought to front.

`nil` *dbid* was not of a window.

Example

```
logWindow =  
    axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)  
; Other interactive code, possibly  
; causing Batch DRC Log window to be covered  
; Uncover the log window:  
axlUIWExpose(logWindow)  
⇒ t
```

- Displays a window using `axlUIViewFileCreate`.
- Interactively moves window behind one or more other windows using the *back* selection of your window manager.
- Calls `axlUIWExpose`.

Window comes to the top above all other windows.

axlUIWClose

```
axlUIWClose(  
    r_window  
)  
⇒ t/nil
```

Description

Closes window *r_window*, if it is open.

Arguments

r_window Window *dbid*.

Value Returned

t Window closed.

nil Window already closed, or *dbid* is not of a window.

Example

```
logWindow =  
    axlUIViewFileCreate( "batch_drc.log" "Batch DRC Log" t)  
; Other interactive code  
;  
    axlUIWClose(logWindow)  
⇒ t
```

- Displays window using `axlUIViewFileCreate`.
- Closes window using `axlUIWClose`.

axlUIWPrint

```
axlUIWPrint(  
    r_window/nil  
    t_formatString  
    [g_arg1 ...]  
)  
⇒ t/nil
```

Description

Prints a message to a window other than the main window. If *r_window* does not have a message line, the message goes to the main window. This function does not buffer messages, but displays them immediately. If the message string does not start with a message class (for example `le`), it is treated as a text (`!t`) message. (See [axlMsgPut](#) on page 614) If `nil`, displays the main window.

Arguments

<i>r_window</i>	Window <i>dbid</i> .
<i>t_formatString</i>	Context message (<code>printf</code> -like) format string.
<i>g_arg1...</i>	Any number of substitution arguments to be printed using <i>t_formatString</i> . Use as you would a C-language <code>printf</code> statement.

Value Returned

<i>t</i>	Message printed to window.
<i>nil</i>	<i>dbid</i> is not of a window.

Example

```
axlUIWPrint(nil "Please enter a value:")  
Please enter a value:  
⇒ t
```

Prints a message in the main window.

axlUIWRedraw

```
axlUIWRedraw(  
    r_window/nil  
)  
⇒ t/nil
```

Description

Redraws the indicated window. If the window *dbid* is *nil*, redraws the main window.

Arguments

r_window Window *dbid* or, if *nil*, the main window.

Value Returned

t Window is redrawn.

nil *dbid* is not of a window.

axlUIWBlock

```
axlUIWBlock(  
    r_window  
)  
⇒ t/nil
```

Description



This function is not compatible with the g_nonBlock = nil option to axlFormCreate. If using this function with axlFormCreate you must set a callback on the g_formAction.

This places a block on the indicated window until it is destroyed. All other windows are disabled. It may be called recursively, unlike the block option in axlFormCreate.

Once you enter a blocking mode you should not bring up a window that is non-blocking. This behavior is not defined and is not supported.

If you block, you should set the block attribute `block` in the Window Placement list `lt_placement` so that the title bar shows it is a blocking window.

If you have a window callback registered you must allow the window to close since the unblock facility unblocks other windows upon close so that the correct window will get the focus after the blocked window is destroyed.

Note: You should set the block symbol option using the `lt_placement` option in the function that creates the window to visually indicate that the window is in blocking mode.

Arguments

`r_window` Window *dbid*.

Value Returned

`t` Success

`nil` Failure (For example, the window is closed or the *dbid* is not of a window).

axlUIEditFile

```
axlUIEditFile(  
    t_filename  
    t_title/nil  
    g_block  
)  
⇒ r_window/t/nil
```

Description

Allows the user to edit a file in an OS independent manner (works under both UNIX and Windows.)

User may override the default editor by setting either the `VISUAL` or `EDITOR` environment variables.

Windows notes

- The default editor is *Notepad*.
- The title bar setting is not supported.

Unix notes

- The default editor is `vi`.
- An additional environment variable, `WINDOW_EDITOR`, allows the user to specify an X-based editor such as `xedit`. The title bar is not supported in this mode.

Note: In blocking mode, the windows of the main program do not repaint until the file editor window exits.

Only `axlUIWClose` supports the `r_window` handle returned by this function.

Arguments

<code>t_filename</code>	Name of file to edit.
<code>t_title</code>	Title bar name, or <code>nil</code> for default title bar.
<code>g_block</code>	Flag specifying blocking mode (<code>t</code>) or non-blocking mode (<code>nil</code>).

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

Value Returned in Non-blocking Mode

r_window Success

nil Failure

Value Returned in Blocking Mode

t Success

nil Failure

axlUIMultipleChoice

```
axlUIMultipleChoice(  
    t_question  
    lt_answers  
)  
⇒ x_answer/nil
```

Description

Displays a dialog box containing a question with a set of two or more answers in a list. The user must choose one of the answers to continue. Returns the chosen answer.

Arguments

<i>t_question</i>	Text of the question for display.
<i>lt_answers</i>	A list of text strings that represent the possible answers.

Value Returned

<i>x_answer</i>	An integer number indicating the answer chosen. This value is zero-based, that is, a zero represents the first answer, a one the second answer, and so on.
<i>nil</i>	An error is detected.

axlUIViewFileScrollTo

```
axlUIViewFileScrollTo(  
    r_windowMsg  
    x_line/nil  
)  
⇒ x_lines/nil
```

Description

Scrolls to a specified line in the file viewer. A value of -1 goes to the end of the viewer.

Note: The number of the line in the view window may not match the number of lines in the file due to line wrapping in the viewer.

Arguments

<i>r_windowMsg</i>	Existing view window.
<i>x_line</i>	Line to scroll: 0 is top of the file, -1 is bottom of the file, -2 returns the number of lines in the viewer.

Value Returned

<i>x_lines</i>	Number of lines in the view window.
<i>nil</i>	No view file window.

Example

```
pm = axlUIViewFileCreate("topology.log" "Topology" nil)  
axlUIViewFileScrollTo(pm -1)
```

- Displays the file topology.log
- Scrolls to the end of the file

axlUIWBeep

```
axlUIWBeep(  
)  
⇒ t
```

Description

Sends an alert to the user, usually a beep.

Arguments

None

Value Returned

None

Example

```
axlUIWBeep( )
```

axlUIWDisableQuit

```
axlUIDisableQuit(  
    o_window  
)  
⇒ t/nil
```

Description

Disables the system menu *Quit* option so the user cannot choose it to close the window.

Arguments

o_window Window handle.

Value Returned

t Window handle is valid.

nil Window handle is invalid.

axlUIWExposeByName

```
axlUIWExposeByName(  
    t_windowName  
)  
⇒ t/nil
```

Description

Finds a window by name and exposes it (raises it to the top of the window stack and restores it to a window state if it is an icon).

You can use the `setwindow` command argument to get PCB Editor window names via scripting. If the window is a form, you get the name by removing the `form.` prefix from its name.

Note: Names of windows may change from release to release.

To raise an item in the control panel, (for example, *Options*,) use the `axlControlRaise()` function.

Arguments

`t_windowName` Window name.

Value Returned

`t` Window is found.

`nil` Window is not found.

axlUIWPerm

```
axlUIWPerm(  
    r_window  
    [t/nil]  
)  
⇒ t/nil
```

Description

Normally forms and other windows close automatically when another database opens. This function allows that default behavior to be overridden.

Notes:

- When you use this function, consider that windows automatically close when a new database opens because the data the windows display may no longer apply to the new database.
- If you do not provide a second argument, returns the current state of the window.

Arguments

<i>r_window</i>	Window id.
<i>t/nil</i>	<i>t</i> - set permanent <i>nil</i> - reset permanent.

Value Returned

<i>t</i>	Window exists.
<i>nil</i>	Window does not exist.

Example 1

```
handle = axlFormCreate('testForm "axlform" nil 'testFormCb, t nil)
axlUIWPerm(handle t)
```

Opens a test form and makes it permanent.

Example 2

```
ret = axlUIWPerm(handle)
```

Tests whether the window is permanent.

axlUIWSetHelpTag

```
axlUIWSetHelpTag(  
    r_window  
    t_tag  
)  
⇒ t/nil
```

Description

Attaches the given help tag to a pre-existing dialog with a port. This function supports subclassing of the help tags, that is, if a help tag is already associated with the dialog, it will not be replaced. This functions adds the new help tag. Adding a new help tag to a pre-existing one is done by concatenating the two with a dot.

For example:

Pre-existing Help Tag:	myOldTag
New Help Tag:	myNewTag
Resulting Help Tag:	myOldTag.myNewTag

Arguments

<i>r_window</i>	Window id.
<i>t_tag</i>	Subclass of the help string.

Value Returned

<i>t</i>	Help tag attached.
<i>nil</i>	Invalid arguments.

axlUIWSetParent

```
axlUIWSetParent(  
    o_childWindow  
    o_parentWindow/nil  
)  
⇒ t/nil
```

Description

Sets the parent of a window. When a window is created, its parent is the main window of the application, which is sufficient for most implementations. To run blocking mode on a form launched from another form, set the child form's parent window to be the launched form.

Setting the parent provides these benefits:

- Allows blocking mode to behave correctly.
- If the parent is closed, then the child is also closed.
- If the parent is iconified, then the child is hidden.
- The child stays on top of its parent in the window stacking order.

Arguments

o_childWindow Child window handle.

o_parentWindow Parent window (if *nil*, then the main window of the application which is normally the default parent.)

Note: A parent and child cannot be the same window.

Value Returned

t Parent is successfully set.

nil Could not set the parent due to an illegal window handle.

axlUIWShow

```
axlUIWShow(  
    r_window/nil  
    s_option  
)  
⇒ t/nil
```

Description

Shows or hides a window depending on the option passed. If the window id passed is `nil`, the function applies to the main window.

Notes:

- Using the `showna` option on a window may make the window active.
- Using the `show` option on a window that is already visible may not make it active.

Arguments

r_window The window id. If `nil`, signifies the main window.

s_option One of the following:

'show	Show and activate the window
'showna	Show but don't activate the window.
'hide	Hide the window.
nil	Show available options.

Value Returned

t Window shown or hidden.

nil Window id not correct or an invalid option given.

axlUIWTimerAdd

```
axlUIWTimerAdd(  
    o_window  
    x_timeout  
    g_oneshot  
    u_callback  
)  
⇒ o_timerId/nil
```

Description

Adds or removes a callback for an interval timer.

This is not a real-time timer. It is synchronous with the processing of window based messages. The actual callback interval may vary. The timer does not go off (and call you back) unless window events for the timer window (*o_window*) are being processed. You must be waiting in a UI related call (for example, *axlEnter**, a blocking *axlFormDisplay*, *axlUIWBlock*, etc.)

To receive callbacks return to the main program message processing. Another window in blocking mode, however, can delay your return to the main program.

You may add properties to the returned *timerId* to store your own data for access in your timer callback as shown:

```
procedure( YourSkillProcedure()  
    ; set up a continuous timer using the main window  
    timerId = axlUIWTimerAdd(nil 2000 nil 'YourTimerCallback)  
  
    timerId->yourData = yourdata  
)  
procedure( YourTimerCallback( window timerId elapsedTime)  
    ;your time period has elapsed. do something.  
)
```

Arguments

o_window

The window the timer is associated with. If *o_window* is *nil*, the timer is associated with the main window.

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

x_timeout Timeout in milliseconds before the timer is triggered and calls your callback procedure. Timeout is not precise because it depends on processing window messages.

g_oneshot Controls how many times the timer triggers. Use one of these values:

- t* - Timer goes off once and automatically removes itself.
- nil* - Timer goes off at the set time interval continuously until it is removed by `axlUIWTimerRemove`.

u_callback Procedure called when the timer goes off. Called with these arguments with its return value ignored:

```
u_callback(  
    o_window  
    o_timerId  
    n_elapsedTime  
)
```

o_window Window you provided to `axlUIWTimerAdd`

o_timerId Timer id which returned by `axlUIWTimerAdd`.

x_elapsedTime Approximate elapsed time in milliseconds since the timer was added.

Value Returned

o_timerId The identifier for the timer. Use this to remove the timer. This return value is subject to garbage collection when it goes out of scope. When the garbage is collected, the timer is removed. Don't count on garbage collection to remove the timer, however, because you do not know when garbage collection will start. If you need a timer that lasts forever, assign this to a global variable.

nil No timer added.

axlUIWTimerRemove

```
axlUIWTimerRemoveSet(  
    o_timerId  
)  
⇒ t/nil
```

Description

Removes a timer added by axlUIWTimerAdd.

Arguments

o_timerId Id returned by axlUIWTimerAdd.

Value Returned

t Timer removed.

nil Timer id invalid.

axlUIWUpdate

```
axlUIWUpdate(  
    r_window/nil  
)  
⇒ t/nil
```

Description

Forces an update of a window. If you made several changes to a window and are not planning on going back to the main loop or doing a SKILL call that requires user interaction, use this call to update a window. You could use this, for example, if you are doing time-consuming processing without returning control to the UI message pump.

Note: This is required for Bristol based code. In other implementations it has no effect.

Arguments

r_window Window id or `nil` if the main window.

Value Returned

t Window updated.

nil Window already closed or invalid window id.

axlUIYesNoCancel

```
axlUIYesNoCancel(  
    t_message  
    [t_title]  
    [s_default]  
)  
⇒ x_result
```

Description

Displays a blocking Yes/No/Cancel dialog box with the prompt message provided.

Arguments

<i>t_message</i>	Message to display.
<i>t_title</i>	Optional. What to put in the title bar of confirm. The default is the program display name.
<i>s_default</i>	Optional. May be either yes, no or cancel to specify default response. The default is yes.

Value Returned

<i>x_result</i>	Number based on the user's choice: 0 for Yes 1 for No 2 for Cancel
-----------------	---

Examples

axlUIDataBrowse

```
axlUIDataBrowse(  
    s_dataType  
    ls_options  
    t_title  
    g_sorted  
    [t_helpTag]  
    [l_callback]  
    [g_args]  
)  
⇒ lg_return
```

Description

Analyzes all objects requested by the caller function, passing each through the caller's callback function. Then puts the objects in a single-selection list.

This list blocks until a user makes a selection. Once the user selects an object, it is passed back to the caller in a list containing two objects: the selected name and, for a database object, the AXL dbid of the object.

Arguments

<i>s_dataType</i>	One of the following: 'NET 'PADSTACK 'PACKAGE_SYMBOL 'DEVICE 'PARTNUMBER 'REFDES 'BOARD_SYMBOL 'FORMAT_SYMBOL 'SHAPE_SYMBOL 'FLASH_SYMBOL 'BRD_TEMPLATE 'SYM_TEMPLATE 'TECH_FILE
-------------------	---

<i>ls_options</i>	List containing at least one of the following: 'RETRIEVE_OBJECT Object selected returns its dbid
-------------------	--

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

'RETRIEVE_NAME	Object selected returns its name
'EXAMINE_DATABASE	Initially look in the database for list of objects
'EXAMINE_LIBRARY	Initially use env PATH variable when looking for list of objects
'DATABASE_FIXED	Read-only checkbox for the database
'LIBRARY_FIXED	Read-only checkbox for files (library)
<i>t_title</i>	Prompt for the title of the dialog
<i>g_sorted</i>	Switch indicating whether or not the list should be sorted
<i>t_helpTag</i>	Help tag for the browser
<i>l_callback</i>	Callback filter function which takes the arguments name, object, and <i>g_arg</i> passed in. Returns t or nil based on whether or not the object is eligible for browsing.
<i>g_arg</i>	Generic argument passed through to <i>l_callback</i> as the third argument.

Value Returned

<i>t_name o_dbid</i>	Selection was made and RETRIEVE_OBJECT used.
<i>t_name nil</i>	Selection was made and RETRIEVE_NAME used.

Examples

```
axlUIDataBrowse('NET '(RETRIEVE_NAME) "hi" t)
axlUIDataBrowse('PADSTACK '(RETRIEVE_NAME) "hi" t)
axlUIDataBrowse('PACKAGE_SYMBOL '(EXAMINE_DATABASE EXAMINE_LIBRARY
    RETRIEVE_NAME)"hi" t)
axlUIDataBrowse('PACKAGE_SYMBOL '(EXAMINE_LIBRARY RETRIEVE_OBJECT) "hi" t)
axlUIDataBrowse('PACKAGE_SYMBOL '(EXAMINE_LIBRARY RETRIEVE_NAME) "hi" t)
axlUIDataBrowse('PARTNUMBER '(RETRIEVE_OBJECT) "Part Number" t)
```

Allegro PCB and Package User Guide: SKILL Reference

User Interface Functions

Form Interface Functions

Overview

This chapter describes the field types and functions you use to create PCB Editor forms (dialogs).

PCB Editor AXL forms support a variety of field types. See [Callback Procedure: formCallback](#) on page 526 and [Using Forms Specification Language](#) on page 457 for a complete description of field types.

See also

[axlFormCreate](#) - open a form

[axlFormCallback](#) - callback model for interaction with user

[axlFormBNFDoc](#) - Backus Naur Form, form file syntax

demos

[axlFormTest](#)

Programming

It is best to look at the two form demo.

- basic controls -- `axlform.il`/`axlform.form`
- grid control - `fgrid.il`/`fgrid.form`

The first step is to create form file. Use `axlFormTest` to insure fields are correctly positioned.

The following procedure is generally used.

1. Open form (`axlFormCreate`)

- 2. Initialize fields (axlFormSetField)**
- 3. Display Form (axlFormDisplay)**
- 4. Interactive with user (axlFormCallback)**
- 5. Close Form (axlFormClose)**



Tip

- ❑ Many users find that it is easier to distribute their program using a form if they embed the form file in their Skill code. In this case use Skill to open a temporary file and print the statements, open for form, then delete the file.
- ❑ Use `axlFormTest("<form file>")` to interactively adjust of fields.
- ❑ You can use "ifdef", "ifndef", and Allegro environment variables (`axlSetVariable`) to control appearance of items in the form file.

Field / Control

Most interaction to the controls are via `axlFormSetField`, `axlFormGetField`, `axlFormSetFieldEditable`, and `axlFormSetFieldVisible`. Certain controls have additional APIs which are noted in the description for the control.

Most controls support setting their background and foreground colors. See `axlColorDoc` and `axlFormColorize` for more information.

Following is a list of fields and their capabilities.

TABSET / TAB

A property sheet control. Provides the ability to organize and nest many controls on multiple tabs.

Unlike other form controls you nest other form controls within TAB/ENDTAB keywords. The size of the tab is control is specified by the FLOC and FSIZE keywords used as part of the TABSET definition. The single option provided to the TAB keyword serves the dual purpose of being both the display name and the tab label name. The TABSET has a single option which is the fieldLabel of the TABSET.

The TABSET has a single option – tabsetDispatch.

When a user picks on a TAB, by default, it is dispatched to the application as the with the fieldLabel set to the name of the tab and the fieldValue as a 't'. With this option we use the fieldLabel defined with the TABSET keyword and the fieldValue as the tab name. In most cases you do not need to handle tab changes in your form dispatch code but when you do each dispatch method has its advantages.

Note: TABSETS cannot be nested.

GROUP

A visible box around other controls. As such, you give it a width, height and optional text. If width or height is 0, we draw the appropriate horizontal or vertical line. Normally the group text is static but you can change it at run-time by assigning a label to the group.

TEXT

Static text, defined in the form file with the keyword "TEXT". The optional second field (use double quotes if more than one word) is any text string that should appear in the field. An optional third field can be used to define a label for run-time control. In addition the label INFO can be used to define the field label and text width.

Mult-line text can be specified by using the FSIZE label with a the height greater than 2. If no FSIZE label is present then a one-line text control is assumed where the field width is specified in the INFO label.

OPTIONS include (form file)

any of:

- bold - text is displayed in bold font
- underline - text is displayed with underline
- border - text is displayed with a sunken border
- prettyprint - make text more read-able using upper/lower case

and one of justification:

- left - left justified (default)
- center - center text in control
- right - right justify text

STRFILLIN

Provides a string entry control. The STRFILLIN keyword takes two required arguments, width of control in characters and string length (which may be a larger or smaller value than the width of the control).

There are three variations of the fillin control.

- single line text
- single line text with a drop-down (use POP keyword).
The dropdown provides the ability to have pre-defined values for the user.
- multi-line text control. Use a FSIZE keyword to indicate field width and height.

INTFILLIN

Similar to a STRFILLIN except input data is checked to be an integer (numbers 0 to 9 and + and -). Use the LONGFILLIN keyword with two arguments; field width and string length.

It only supports variations 1 and 2 of STRFILLIN.

It also supports a minimum and maximum data verification. This can be done via the form file with the MIN and MAX keywords or at run-time via `axlFormSetFieldLimits`.

INTSLidebar

This is a special version of the INTFILLIN, it provides an up/down control to the right of the field that allows the user to change the value using the mouse. You should use MIN/MAX settings to limit the allowed value.

REALFILLIN

Similar to INTFILLIN except supports floating point numbers. Edit checks are done to only allow [0 to 9 .+-]. If addition to min/max support you can also provide number of decimals via the DECIMAL keyword or at run-time via `axlFormSetDecimal`.

MENUBUTTON

Provides a button control. Buttons are stateless. The MENUBUTTON keyword takes two options; width and height.

A button has one option – multiline.

If button text cannot fit on one line wrap it. Otherwise text is centered and restricted to a single line.

A button can have a popup by inserting the "POP" label.

With no popup pressing the button dispatches a value of 1. If it is a button with a popup then the dispatch is the dispatch entry of the popup.

Standards:

- use "..." if button brings up a file browser
- append "..." to text of button if button brings up another window
- use these labels for:
 - close - to Close dialog without
 - done/ok - to store changes and close dialog
 - cancel - to cancel dialog without making any changes
 - help - The is reserved for cdsdoc help
 - print - do not use (will get changed to Help).

CHECKLIST

Provides a check box control (on/off). Two variants are supported:

- a checkbox
- a radiobox

For both types the CHECKLIST control takes an argument for the text that should appear to the right of the checkbox.

A radiobox allows you to several checkboxes to be grouped together. The form package insures only one radiobox be set. To enable a radio grouping provide a common text string as a third argument to the CHECKLIST keyword. An idiosyncrasy of a radiobox is that you will be dispatched for both the field being unset and also for the field being set.

ENUM (*sometimes called combo box*)

Provides a dropdown to present the user a fixed set of choices. The dropdown can either be pre-defined in the form file via the POPUP keyword or at run-time with `axlFormBuildPopup`. Even if you choose to define the popup at run-time, you must provide a POPUP placeholder in the form file.

POPUP entries are in the form of display/dispatch pairs. Your setting and dispatching of this field must be via the dispatch item of the popup (you can always make both the same). This technique allows you to isolate what is displayed to the user from what your software uses. The special case of nil as a value to `axlFormSetField` will blank the control.

Two forms of ENUM field are supported, the default is single line always has the dropdown hidden until the user requests it. In this case only define the ENUMSET with the width parameter. A multi-line version is available where the drop-down is always displayed. To enable the multi-line version specify both the width and height in ENUMSET keyword.



Tip
FILLIN fields also offer ENUM capability, see below.

OPTIONS include (form file)

`prettyprint` - make text more read-able using upper/lower case.

`ownerdrawn` - provided to support color swatches next.

to subclass names. See `axlSubclassFormPopup`.

`dispatchsame` - Normally if user selects same entry that

is currently shown it will not dispatch.

LIST

A list box is a control that displays multiple items. If the list box is not large enough to display all the list box items at once, the list box provides the required horizontal or vertical scroll bar.

We support two list box types; single (default) and multi-selection. You define a multi-select box in form file with a "OPTIONS multiselect" List boxes have a width and height specified by the the second and third options to the LIST keyword. The first option to the LIST keyword is ignored and should always be an empty string ("").

List box options are:

`SORT` - alphabetical sort.

ALPHANUMSORT - takes in account trailing numbers so a NET2 appears before a NET10 in the list.

PRETTYPRINT - case is ignored and items are reformatted for readability.

Special APIs for list controls are: axlFormListOptions, axlFormListDeleteAll, axlFormListSelect, axlFormListGetItem, axlFormListAddItem, axlFormListDeleteItem, axlFormListGetSelCount, axlFormListGetSelItems, axlFormListSelAll.

For best performance in loading large lists consider passing a list of items to axlFormSetField.

THUMBNAIL

Provides a rectangular area for bitmaps or simple drawings. You must provide a FSIZE keyword to specify the area occupied by the thumbnail.

In bitmap mode, you can provide a bitmap as an argument to the THUMBNAIL keyword or at run time as a file to axlFormSetField. In either case, BMPPATH and a .bmp extension is used to locate the bitmap file. The bitmap should be 256 colors or less.

For bitmaps one OPTION is supported:

stretch - draw bitmap to fill space provided. Default is to center bitmap in the thumbnail region.

In the drawing mode you use the APIs provided by axlGRPDOC to perform simple graphics drawing.

TREEVIEW

Provides a hierachial tree selector. See axlFormTreeViewSet.

GRID

This provides a simple spreadsheet like control. See axlFormGridDoc for more info.

COLOR

Provides a COLOR swatch. Can be used to indicate status (for example: red, yellow, green). The size of the color swatch is controlled by a width and height option the COLOR keyword.

Add the INFO_ONLY keyword to have a read-only color swatch. Without INFO_ONLY the color swatch provides CHECKBOX like functionality via its up/down appearance.

With COLOR swatches you can use predefine colors or Allegro database colors. See axlColorDoc.

TRACKBAR

Provides a slider bar for setting integer values. The TRACKBAR keyword takes both a width and height and the bar may be either horizontal or vertical.

It is important to set both a minimum and maximum integer value. This can be done from the form file with the MIN and MAX keywords or at run-time by axlFormSetFieldLimits.

PROGRESS

Provides a progress bar usually used to indicate status of time consuming operations. For setting options to the progress meter pass of list of 3 items to axlFormSetField which are (<step value> <number of steps> <initial position>). A subsequent nil passed to axlFormSetField will step the meter by the <step value>.

PROGRESS keyword provides for both a width and height of the bar. Bar should be horizontal.

You get information from the user using forms that support the following modes:

Table 11-1 Form Modes

Form Mode	Description
Blocking with no callback	<p>Easy to program. Limited to user interaction, such as checking that the information entered for each field uses syntax acceptable to the form's package. Your program calls <code>axlUIWBlock</code> after displaying the form. The user can close a form that has the standard <i>OK</i> or <i>Cancel</i> button.</p> <p>After <i>OK</i> or <i>Cancel</i> is selected, <code>axlUIBlock</code> returns allowing you to query field values using <code>axlFormGetField</code>.</p> <p>Note: Use this programming model only with simple forms.</p>
Blocking with callback	<p>Prevents use of PCB Editor until the user enters information in the dialog. The form callback you provide lets your interactive program accept the data entered.</p>
Callback with no blocking	<p>Works like many native PCB Editor forms. The user can work with both the form and other parts of PCB Editor.</p> <p>With PCB Editor database transactions, the programming is more complex. You can use transactions while the form is open by declaring your command interactive. You end your command when another PCB Editor command starts by using <code>axlEvent</code>.</p>

Form Mode	Description
Options form	PCB Editor window to the left of the canvas. The options (ministatus) form is non-blocking and restricted to the Options panel size. See <code>axlMiniStatusLoad</code> for details.

Do not attempt to set the `Button` field (except `Done`, `Cancel` and `Help`), as it is designed to initiate actions. Consequently, having buttons in a form without a callback function registered renders those buttons useless.

Note: AXL-SKILL does not support the short fields and variable tiles which are part of the PCB Editor core form package.

You can set background and foreground color on many form fields. For more information, see [axlFormColorize](#) on page 532. For information on color specific to grids, see [Using Grids](#) on page 470.

Examples

These examples, especially the basic one, help you understand how the forms package works:

- | | |
|--------|--|
| basic | Demonstrates basic form capabilities. |
| grid | Demonstrates grid control capabilities. |
| wizard | Demonstrates use of a form in Wizard mode. |

Use the examples located in `<cdsroot>/share/pcb/examples/form` as follows:

1. Copy all the files from one of the directories to your computer.
2. Start PCB Editor.
3. From the PCB Editor command line, change to the directory to which you copied the files as shown:
`cd <directory>`
4. Load the SKILL file in the directory.

Note: The SKILL file has the `.il` extension.

```
skill load "<filename>"
```

5. Start the demo by typing on the PCB Editor command line as shown:

For basic demo:

```
skill formtest
```

For grid demo:

```
skill gridtest
```

6. Examine the SKILL code and form file.



Tip
Setting the PCB Editor environment variable TELSKILL opens a SKILL interpreter window that is more flexible than the PCB Editor command area. On UNIX, if you set this variable before starting the tool then the SKILL type-in area is the X terminal you used to start PCB Editor. See the enved tool to configure the width and height of the window.

Using Forms Specification Language

Backus Naur Form (BNF) is a formal notation used to describe the syntax of a language. Form File Language Description is the BNF grammar for the Forms Specification Language. Forms features in new versions are not backwards compatible.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

The following table shows the conventions used in the form file grammar:

Convention	Description
[]	Optional
{ }	May repeat one or more times
< >	Supplied by the user
	Choose one or the other
:	Definition of a token
CAPS	Items in caps are keywords

The BNF format definition follows.

```
BNF:
form:
    FILE_TYPE=FORM_DEFN VERSION=2
    FORM [form_options]
    formtype
    PORT w h
    HEADER "text"
    form_header
    {tile_def}
    ENDFORM

formtype:      FIXED | VARIABLE
              - FIXED forms have one unlabeled TILE stanza
              - VARIABLE forms have one or more label TILE stanzas
              - Skill only supports FIXED form types.

PORT:
        - Width and height of the form. Height is ignored for fixed forms
          which auto-calculate required height. Width must be in character
          units.

HEADER:
        - Initial string used in the title bar of the form. This may be
          overridden by the application.

form_header:
        [{default_button_def}]
        [{popup_def}]
        [{message_def}]

default_button_def:
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
DEFAULT <label>
  - Sets the default button to be <label>. If not present, the form
    sets the default button to be one of the following: ok (done),
    close, or cancel.
  - Label must be of type MENU BUTTON.

popup_def:
  POPUP <><popupLabel>> {"<display>","<dispatch>"}.
  - Popups may be continued over several lines by using the backslash
    (\) as the last character on a line.

message_def:
  MESSAGE messageLabel messagePriority "text"

form_options:
  [TOOLWINDOW]
  - This makes a form a toolwindow which is a floating toolbar. It
    is typically used as a narrow temp window to display readouts.

  [FIXED_FONT]
  - By default, forms use a variable width font. This option sets the
    form to use a fixed font. Allegro/Apd use mostly variable width
    while SPECCTRAQuest and SigXP use fixed width fonts.

  [AUTOGREYTEXT]
  - When a fillin or enum control is greyed, grey static text to the
    left of it.

  [UNIXHGT]
  - Works around a problem with Mainsoft in 15.0 where a button is
    sandwiched vertically between 2 combo/fillin controls. The
    button then overlaps these controls. This adds extra line spacing
    to avoid this. You should only use this option as a last resort.
    In a future release, it may be treated as a Nop. On Windows, this
    is ignored.

tile_def:
  TILE [<titleLabel>]
  [TPANEL tileType]
  [{text_def}]
  [{group_def}]
  [{field_def}]
  [{button_def}]
  [{grid_def}]
  [{glex_def}]
  END TILE

tabset_def:
  TABSET [label]
  [OPTIONS tabsetOptions]
  FLOC x y
  FSIZE w h
  {tab_def}
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
ENDTABSET

tab_def:
    TAB "<display>" [<label>]
    [{text_def}]
    [{group_def}]
    [{field_def}]
    [{grid_def}]
    ENDTAB

text_def:
    TEXT "display" [label]
    FLOC x y
    [FSIZE w h]
    text_type
    [OPTIONS textOptions]
    ENDTEXT

text_type:
    [INFO label w] |
    [THUMBNAIL [<bitmapFile>|#<resource>] ]

group_def:
    GROUP "display" [label]
    FLOC x y
    [INFO label]
    FSIZE w h
    ENDGROUP

field_def:
    FIELD label
    FLOC x y
    [FSIZE w h]
    field_type
    field_options
    ENDFIELD

button_def:
    FIELD label
    FLOC x y
    [FSIZE w h]
    MENUBUTTON "display" w h
    button_options
    ENDFIELD

grid_def:
    GRID fieldName
    FLOC x y
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
FSIZE w h
[OPTIONS INFO | HLINES | VLINES | USERSIZE ]
[POP "<popupName>" ]

[GHEAD TOP|SIDE]
[HEADSIZE h|w]
[OPTION 3D|NUMBER]
[POP "<popupName>" ]
[ENDGRID]
ENDGRID

field_type:
    REALFILLIN w fieldLength |
    LONGFILLIN w fieldLength |
    STRFILLIN w fieldLength |
    INTSLIDEBAR w fieldLength |
    ENUMSET w [h] |
    CHECKLIST "display" ["radioLabel"] |
    LIST "" w h |
    TREEVIEW w h |
    COLOR w h |
    THUMBNAIL [<bitmapFile>|#<resource>] |
    PROGRESS w h
    TRACKBAR w h

field_options:
    [INFO_ONLY]
        - Sets field to be read-only

    [POP "<popupName>" ]
        - Assigns a popup with the field.
        - A POPUP definition by the same name should exist.
        - Supported by field_types: xxxFILLIN, INTSLIDEBAR, MENUBUTTON, and
          ENUMSET.

    [MIN <value>]
    [MAX <value>]
        - Assigns a min and/or max value for the field.
        - Both supported by field types: LONGFILLIN, INTSLIDEBAR,
          REALFILLIN.
        - Value either an integer or floating point number.
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
[DECIMAL <accuracy>]
  - Assigns a floating min and/or max value for the field.
  - Assigns the number of decimal places the field has (default is 2)
  - Both supported by field_types: REALFILLIN

[VALUE "<display>"]
  - Initial field value.
  - Supported by field_types: xxxFILLIN

[SORT]
  - Alphanumeric sorted list (default order of creation)
  - Supported by field_type: LIST

[OPTIONS dispatchsame]
  - For enumset fields only
  - If present, will dispatch to application drop-down selection even if the same as current. By default, the form's package filters out any user selection if it is the same as what is currently displayed.

[OPTIONS prettyprint]
  - For enumset fields only.
  - Displays contents of ENUM field in a visually pleasing way.

[OPTIONS ownerdrawn]
  - For enumset fields only.
  - Used to display color swatches in an ENUM field. See axlFormBuildPopup.

x:
y:
w:
h:
  - Display geometry (integers)
  - All field, group and text locations are relative to the start of the tile they belong or to the start of the form in the case of FIXED forms.
  - x and h are in CHARHEIGHT/2 units.
  - y and w are in CHARWIDTH units.

button_options:
  [MULTILINE]
    - Wraps button text to multiple lines if text string is too long for a single line.

dispatch:
  - String that is dispatched to the code.

display:
  - String that is shown to the user.
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

`bitmapFile:`

- Name of a bmp file. Finds the file using BITMAPPATH

`resource:`

- Integer resource id (bitmap must be bound in executable via the resource file). '#' indicates it is a resource id.
- Not supported in AXL forms.

`fieldLength:`

- Maximum width of field. Field scrolls if larger than the field display width.

`label:`

- Name used to access a field from code. All fields should have unique names.
- Labels should be lower case.

`messageLabel:`

- Name used to allow code to refer to messages.
- Case insensitive.

`messagePriority:`

- Message priority 0 - (not in journal file), 1 - information, 2 - warning, 3 - error, 4 - fatal (display in message box)

`radioLabel:`

- Name used to associate several CHECKLIST fields as a radio button set. All check fields should be given the same radioLabel.
- Should use lower case.

`textOptions:`

[RIGHT | CENTER | BORDER | BOLD | UNDERLINE]

- TEXT/INFO field type
 - text justification, default is left
 - BORDER: draw border around text
- [STRETCH]
- THUMBNAIL field type
 - Stretch bitmap to fit thumbnail rectangle, default is center bitmap.

`tabsetOptions:`

[tabsetDispatch]

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

- By default, tabs dispatch individual tabs as separate events. This is not always convenient for certain programming styles. This changes the dispatch mode to be upon the tabset where a selection of a tab causes the event:

```
field=tabsetLabel value=tabLabel
```

The default is:

```
field=tabLabel value=t
```

Script record/play remains based upon tab in either mode.

tileLabel:

- Name used to allow code to refer to this tile.
- Should use lower case.
- Only applies to VARIABLE forms.
- Not supported with AXL forms.

tileType [0|1|2]

- 0 top tile, 1 scroll tile, 2 bottom tile
- Only applies to VARIABLE FORMS.
- Region where tile will be instantiated. Forms have the following regions: top, bottom, and scroll (middle).
- Not supported with AXL forms.

flex_def: Rule based control sizing upon form resize (see axlFormFlex)

```
[FLEXMODE <autorule>]  
[FLEX <label> fx fy fw fh]
```

FLEXMODE <autoRule>

FLEX fx fy fw fz

- see axlFormFlexDoc

autorule: - Generic sizing placement rule.

fx:

fh:

- Floating value between 0 and 1.0



Follow these rules when using BNF format:

- FILE_TYPE line must always appear as the first line of the form file in the format shown.
- Form files must have a .form extension.
- There may only be one FORM in a form file.
- There must be one and only one TILE definition in a FIXED form file. <tileLabel> and TPANEL are not required.
- Unless otherwise noted, character limits are as follows:
 - labels - 128
 - title - 1024
 - display - 128 except for xxxFILLIN types which are 1024
- Additional items may appear in existing form files (FGROUP) but they are obsolete and are ignored by the form parser. REALMIN and REALMAX are obsolete and replaced by MIN and MAX respectively. They will still be supported and are mapped to MIN and MAX.
- For *grid_def*, two headers (side and top) are maximum.
- FSIZE - Most controls determine the size from the text string.
You must provide FSIZE for GROUP, GRID, TREEVIEW and LIST controls. For TEXT controls, if FSIZE is provided, it overrides the width calculated by the text length and, if present, the INFO width. If using the INFO line, put the FSIZE line after it.
- Both TEXT and GROUP support the optional label on their definition line. This was added as a convenience in supporting FLEX capability. If the application wishes to dynamically modify the text, the INFO keyword is normally used. When both are present, the INFO keyword takes precedence.
- If the optional label for TABS is not provided, the field display name is used. Any spaces within the field display name are replaced by underscores ("_").
- The height ([*h*]) for ENUMSET is optional. When not set (the default), the drop-down is only presented under user control. When height is greater than 1, the drop-down is always visible (Microsoft SIMPLE drop-down). Only use this feature in forms that can afford the space consumed by the drop-down.

The forming syntaxes are NOT supported by the form editor.

This syntax is supported and may be placed anywhere in the form file to support conditional processing of the form file:

```
#ifdef <variable>
{}

{ #elseif <variable>
}
{     #else
{}     }
```

Moving and Sizing Form Controls During Form Resizing

You can move and size controls within a form based on rules described in the form file. Rules may either be general (**FLEXMODE**) or specific to a single control (**FLEX.**) Flex adjusting of the controls is adjusting the form larger than its base size. Sizing the form smaller than the base size disables flex sizing.

Controls are divided into the following classes:

- Containers
Containers can have other controls as members, including other containers. To be a container member is automatic; the control's *xy* location must be within the container. Container controls of the form are TABSETS and GROUPS.
- All others, including containers

All controls except TABS, which are locked to their TABSET, may be moved when a form is resized. Sizing width or height is control dependent as shown:

Table 11-2 Controls - Resizing Options

Control	Resizing Options
REALFILLIN	width
LONGFILLIN	width
STRFILLIN	width
INTSLIDE BAR	width

Table 11-2 Controls - Resizing Options, continued

Control	Resizing Options
ENUMSET	width
PROGRESS	width
TRACKBAR	width
LIST	width and height
GRID	width and height
TREEVIEW	width and height
THUMBNAIL	width and height
GROUP	width and height
TABSET	width and height
<others>	no change in size

Using Global Modes or FLEXMODE

FLEXMODE represents the general rules that apply to all controls in the form except those with specific overrides (FLEX). Only a single FLEXMODE is supported per form. The last encountered in the form file is used. The following rules are supported:

- EdgeGravity
All controls have an affinity to the closest edge of their immediate container. Exceptions are: <xxx>FILLIN and INTSLIDE BAR controls. The edge gravity, for these, is based upon a TEXT control positioned to the left of the control.
- EdgeGravityOne
Similar to EdgeGravity except that controls are only locked to the right or bottom edge, but not both. The closest edge is used.
- StandButtons
Only effects button controls. Uses the same logic as EdgeGravityOne.

Managing Sizing and Movement of Individual Controls

You use the FLEX parameter to manage the sizing and movement of individual controls as shown:

```
FLEX fx fy fw fh
```

The **FLEX** parameter overrides any **FLEXMODE** in effect for that control, and is based upon parameters (f_x , f_y , f_w , f_h). These values, which are floating point numbers between 0.0 and 1.0, control the fraction of the change in container size that the control should move or change in size:

fx and fy Parameters

- | | |
|---|--|
| 0 | Control remains locked to the left or top edge of its container. |
| 1 | Control remains locked to the right or bottom edge of its container. |

fw and fh Parameters

- | | |
|---|--|
| 0 | Control is not resized. |
| 1 | Control is resized in width or height based upon the size change of its container. |

A container's position and size effect the container's member controls. Containers are hierarchical. Make sure the container of the control also has a **FLEX** constraint. The sum of the width and height of the immediate controls of a container should not be greater than 1 to prevent overlapping. TABSETS are slightly different since sizing of their member controls is also based on the TAB they belong to.



It is possible to create FLEX constraints that result in overlapping controls. FLEX does not protect against this.

FLEX Restrictions

- The form must be **FIXED**.
- While **FLEX** rules may appear anywhere in the form file, they should be grouped together immediately before the **<ENDTILE>**
- Range errors for **FLEX** option or applying width or height to controls not supporting them are silently ignored.

Example 1

```
FLEXMODE standbuttons  
FLEX list 0 0 1 1
```

Simple list-based form with buttons (label of LIST is list.) The list gets all of form sizing.

Example 2

```
FLEXMODE EdgeGravity  
FLEX a 0 0 0.33 1  
FLEX b 0.33 0 0.67 1  
FLEX c 0.67 0 1 1
```

Form containing 3 lists (a, b, and c) positioned equally across the form. Each list gets the total change in height, but shares in the increase in form width. Thus, if the form changes width, each control gets 1/3 of this change. Since the list's widths change, the list must move to the right.

Example 3

```
FLEX l1 0 0 1 0.5  
FLEX g1 0 0.5 1 0.5  
FLEX l2 0 0 1 1
```

Form has a group (g1) containing a list (l2). These are at the bottom of another list (l1). Both lists share in any change of the form size. The second list (l2) is a member of the group container (g1), so it moves if the group moves (0 for *y*) and it gets all of the group resizing (*h* is 1).

Example 4

```
FLEX g1 1 1 0 0  
FLEX l1 0 0 1 1
```

Form has a group (g1) with a list member (l1), but the list doesn't resize because the list is a member of the group which has 0:0 sizing. Though the list has 1:1 sizing, it never changes in size because its container never changes in size. Both the group and its member list move because the group has a 1:1 *x/y* factor.

Example 5

```
FLEX t1 0 0 1 1  
FLEX l1 0 0 1 1  
FLEX l2 0 0 1 1
```

Form is a tabset (t1) with 2 tabs. Each tab controls a list (l1 and l2) that accommodates the maximum change in the form size.

- Use `axlFormTest(<formname>)` to experiment with your form.

Using Grids

Grids offer tabular support and the following features:

- Optional side and top headers
- Several data types on a per column basis: Text (info), Checkbox with optional text, Enum (Drop-drop) and Fillin (text box with built-in types: string, integer, and real.)
- Row and column indexing which is 1-based

Grids have the following limits:

- Maximum of 200 columns
- Maximum rows of 1,000,000
- Maximum field string length per column of 256 characters
- Column creation only at grid initialization time.

Form File Support for Grids

The following defines the form file structure relating to grids.

GRID

Standard items

FLOC - x, y location
FSIZE- width and height including headers if used
POP - Optional right button popup for body. Also requires application to set the GEVENT_RIGHTPOPUP option.

OPTIONS:

INFO - Entire grid is info-only even if it contains typeable fields
HLINES- Draw horizontal lines between columns
VLINES- Draw vertical lines between rows
USERSIZE- Allow user to resize columns.

HEADERS (GHEAD)

- Specified within GRID section.
- TOP and SIDE header (only one per type allowed in a grid)

HEADSIZE- Height (TOP) or width (SIDE) for the header.

OPTIONS:

3D - Display raised.
NUMBER- For side header, display row number if application does not provide text.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

POP - Optional right mouse button popup. One per header. Requires application to set GEVENT_RIGHTPOPUP for the header.

Programming Support for Grids

The following Grid APIs are available:

axlFormGridInsertCol	Insert a column.
axlFormGridInsertRows	Insert one or more rows.
axlFormGridDeleteRows	Delete one or more rows.
axlFormGridEvents	Set grid events.
axlFormGridOptions	Miscellaneous grid options.
axlFormGridNewCell	Obtain structure for setting a cell.
axlIsGridCellType	Is item a cell data type.
axlFormGridSetBatch	For setting multiple cells.
axlFormGridGetCell	For getting cell data.
axlFormGridBatch	Used with axlFormGridSetBatch
axlFormGridUpdate	Update display after changes.
make_formGridCol	For defstruct formGridCol
copy_formGridCol	For defstruct formGridCol

In addition, the following standard form APIs may be used:

<code>axlFormSetFieldVisible</code>	Set grid visibility
<code>axlFormIsFieldVisible</code>	Is field visible
<code>axlFormSetFieldEditable</code>	Set grid editability
<code>axlFormIsFieldEditable</code>	Is field editable
<code>axlFormBuildPopup</code>	Change a popup
<code>axlFormSetField</code>	Set individual cell.
<code>axlFormRestoreField</code>	Restore last cell changed. Restore supports undoing last <i>change</i> event. Adding, deleting, or right mouse event reset restore.

Data Structures

<code>r_cell</code>	User data type for cell update (see axlFormGridNewCell on page 550)
<code>r_formGridCol</code>	Defstruct to describe column (see axlFormGridInsertCol on page 544)

Column Field Types

Grids support the assignment of data types by column. You may change an editable cell into a read-only cell by assigning it a `s_noEdit` or `s_invisible` attribute. See `axlFormGridInsertCol` for a complete description of column attributes and `axlFormGridSetBatch` for a discussion of cell attributes.

<code>TEXT</code>	Column is composed of display only text.
<code>STRING</code>	Column supports editable text. See edit-combo.
<code>LONG</code>	Column supports numeric data entry cells. See edit-combo.
<code>REAL</code>	Column supports numeric floating point entry cells. See edit-combo.

ENUMSET	Column supports combo-box (drop-down) cells. Must have a popup attribute on the column.
CHECKITEM	Column has checkbox cells with optional text.
EDIT-COMBO	By assigning a popup attribute at the column and/or at the cell level, you can change STRING, LONG, and REAL types to support the original text editing field with the addition of a drop-down.

Initializing the Grid

Once a grid is defined in the form file, you can initialize the grid as follows:

1. Create required columns using `axlFormGridInsertCol`
2. Create initial set of rows using `axlFormGridInsertRows`
3. Create initial grid cells and headers using `axlFormGridSetBatch`, then on callback, use:
 - a. `axlFormGridNewCell`
 - b. `axlFormGridSetBatch`
4. Set event filters using `axlFormGridOptions`.
5. Display the grid using `axlFormGridUpdate`.

See `grid.il` and `grid.form` for a programming example. You can find these in the AXL Shareware area:

`<CDS_INST_DIR>/share pcb/etc/skill/examples/ui`

Dispatching Events

Unlike other form controls, an application can specify what events are dispatched. You control this using the `axlFormGridEvents` API which documents the usage. Also, the form callback structure has new fields for grids (see [axlFormGridEvents](#) on page 539.)

By default, you create a grid with the 'rowselect' enabled which is typically appropriate for a multi-column table.

Using Scripting with Grid Controls

Unlike most other form controls where the programmer needs no concern over scripting, grid programmers should address scripting. By default, the grid uses the event type and row/column number for scripting. Depending on your application, this may create scripts that do not replay given different starting data. Grids support assigning script labels to rows, to columns, and on a per cell basis.

You label by setting the *scriptLabel* attribute from the application code with the `axlFormGridInsertCol` function for a column or the `axlFormGridNewCell` function for a row, column, or per cell basis. You can also change this dynamically. Note that (`row=0, col=n`) sets the *scriptLabel* for the column using `axlFormGridNewCell` and (`row=n, col=0`) allows setting for row script labels.

The grid script line format extends upon the standard form scripting as shown:

```
FORM <formname> [<titleLabel>] <fieldLabel> <event> <glabel> [<value>]
```

where

```
FORM <formname> [<titleLabel>] <fieldLabel>
- standard form script form fieldLabel is the grid label
<event> is the grid event. Grid events include:
    rowselect      := GEVENT_ROWSELECT
    cellselect     := GEVENT_CELLSELECT
    change         := GEVENT_CELLCHANGE
    rpopup         := GEVENT_RIGHTPOPUP
    rprepopup      := GEVENT_RIGHTPOPUPPRE
    lpopup         := GEVENT_LEFTPOPUPPRE
<glabel> label corresponds to the location in the grid the event
          occurred.
[<value>] optional value depending upon event.
```

Depending on the event, the rest of the script line appears as follows:

```
rowselect  <glabel:=row>
cellselect <glabel:=cell>
change     <glabel:=cell> <value>
rpopup     <glabel:=cell> <popup value>
rprepopup  <glabel:=cell>
lpopup     <glabel:=cell>
```

The `glabel` has several format options depending on the event:

- | | |
|-------------------|---|
| <code>row</code> | If the row has a <code>scriptLabel</code> , it is used, otherwise the row number is used. |
| <code>cell</code> | If the cell has a label, that is used. If the cell does not have a label, the row and /or column labels are used. If either the row or column does not have labels, the row and/or column number is used. |

When you set a `scriptLabel` to `row`, `col`, or `cell`, the following character set is enforced: case insensitive, no white space or comma or \$. Labels with these characters are replaced by an underscore (_). You may use pure numeric strings, but if you do not label everything, scripts may fall back and use the row/grid number to resolve a number not found as a script label string.

Notes

- If you use `row` and `col` as the `glabel`, use a comma (,) to delineate between the row and column name and number.
- Do not turn on events that you do not plan to process since scripts record them. For instance, if you only process on `rowselect` (no editable cells), then only enable `rowselect`. As a side benefit, you do not have to label columns or cells since row label is sufficient.
- If you use a row and/or column heading, you may use that for assigning `scriptLabels`.

Examples

- If grids replace the text parameter form, you need not label the columns. A column number is sufficient. You can label the columns for script readability. This application does not require cell labeling.
- If grids replace the color form for certain color grids, like stackup, you would need to label each cell. Each class grouped in the stackup grid is not row consistent. For example, depending on design, subclasses are not the same going across the rows. Other groupings require labeling on class for `col` and subclass for `row` since it is orthogonal.

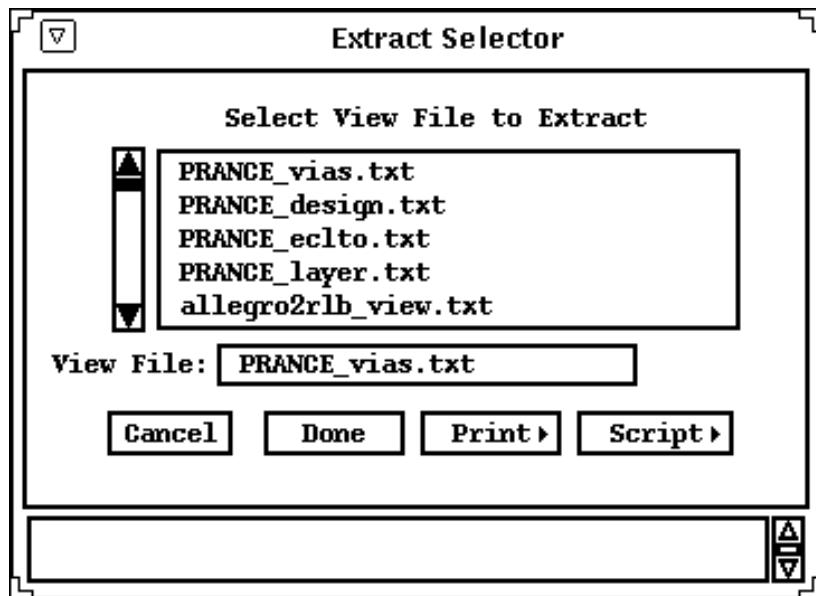
See [Using Grids](#) on page 470 for a grid overview.

Headers

You can set column (top) headers either using `axlFormGridInsertCol` at column creation time, or using `axlFormGridSetBatch` if you need to change the header using row number 0.

Row (side) headers default to automatic run numbers with this option set in the form file. Using `axlFormGridSetBatch`, you can set the text for individual rows using col number 0.

AXL Forms: Example 1



```
FILE_TYPE=FORM_DEFN VERSION=2
FORM
FIXED
PORT 50 11
HEADER "Extract Selector"
TILE
TEXT "Select View File to Extract"
TLOC 12 1
ENDTEXT
TEXT "View File:"
TLOC 1 12
ENDTEXT
FIELD view_file
FLOC 12 12
STRFILLIN 24 24
ENDFIELD
FIELD file_list
FLOC 5 3
LIST "" 40 5
ENDFIELD
FIELD cancel
FLOC 5 15
MENUBUTTON "Cancel" 8 3
ENDFIELD
FIELD done
FLOC 15 15
MENUBUTTON "Done" 9 3
ENDFIELD
FIELD print
FLOC 25 15
MENUBUTTON "Print" 9 3
ENDFIELD
FIELD script
FLOC 35 15
MENUBUTTON "Script" 11 3
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
ENDFIELD  
ENDTILE  
ENDFORM
```

- Uses a form file (expected to be in the current directory) that can display a selection list.
- Gets the list of available extract definition (view) files pointed to by the TEXTPATH environment variable.
- Displays the list in the form.

The user can then select any filename listed, and the name displays in the *View File* field.

Selecting the *Done* button causes the form to call `axlExtractToFile` with the selected extract filename as the view file, and `myextract.dat` as the extract output filename, and closes the form. Selecting *Cancel* cancels the command and closes the form.

The form file has `FIELD` definitions for the selection list, the *View File* field, and each of the buttons (*Cancel*, *Done*, *Print* and *Script*).

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

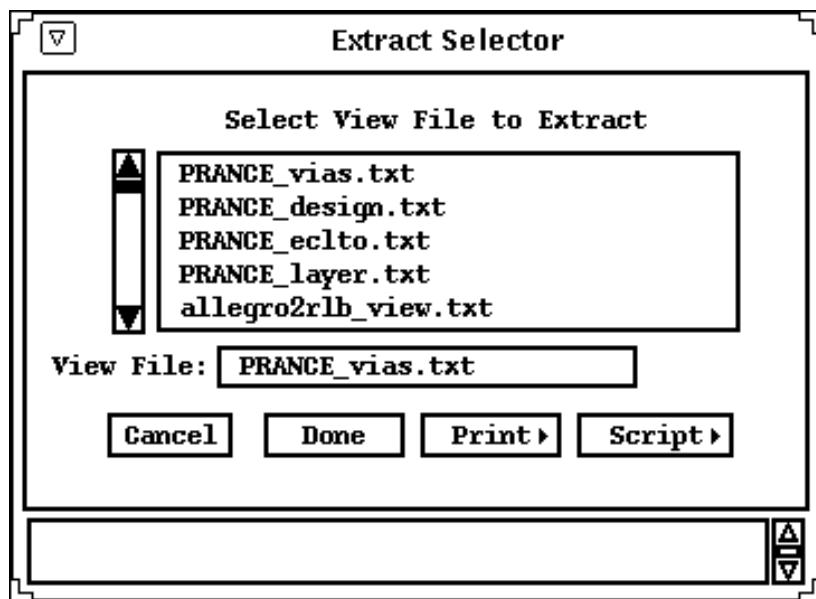
```
; myExtractViews.il
;           -- Displays a form with a selection list of
;           the available extract definition files
;           -- Lets the user select any of the files on
;           the list as the "View file"
;           -- Starts Allegro extract process with the
;           user-selected View file when
;           the user picks Done from the form.

; Function to extract user selected view to the output file.
(defun myExtractViews (viewFile outFile)
  axlExtractToFile( viewFile outFile)
); defun myExtractViews
; Function to start the view extraction
(defun _extract ()
  myExtractViews(
  buildString(list(cadr(parseString(
    axlGetVariable("TEXTPATH")))) selectedFile) "/"))
  "myextract.dat")
); defun _extract
; Form callback function to respond
(defun _formAction (form)
  (case form->curField
    ("done"
      (axlFormClose form)
      (axlCancelEnterFun)
      (_extract)
      t)
    ("cancel"
      (axlFormClose form)
      (axlCancelEnterFun)
      nil)
    ("view_file"
      (if form->curValue
        (progn
          ; Accept user input only if on list
          if(member( form->curValue fileList)
            then axlFormSetField( form
              "view_file" form->curValue)
            else axlFormRestoreField(
              form "view_file")))))
      t)
    ("file_list"
      (axlFormSetField form "view_file"
        form->curValue)
      selectedFile = form->curValue
      t)); case
); defun _formAction
; User-callable function to set up and
;   display the Extract Selector form
(defun myExtract ()
  fileList = (cdr (cdr (getDirFiles
    cadr( parseString( axlGetVariable("TEXTPATH"))))))
  form = axlFormCreate( (gensym)
    "extract_selector.form" '("E" "OUTER")
    '_formAction t)
  axlFormTitle( form "Extract Selector")
  axlFormSetField( form "view_file" (car fileList))
  selectedFile = (car fileList)
  foreach( fileName fileList
    axlFormSetField( form "file_list" fileName))
  axlFormDisplay( form)
); defun myExtract
```

- Creates a form named *form* with the callback function *_formAction* that analyzes user action stored in *form->curField* and responds appropriately.

- Loads the example AXL program shown.
- Enters the command `myExtract()`.

SKILL displays the **Extract Selector** form, as specified in the form file `extract_selector.form` that this code created when it first loaded. This is a non-blocking form—you can enter other SKILL and PCB Editor commands while the form displays.



The program shows how to analyze the user selection when control passes to the callback function `_formAction`. Name of the field selected by the user is in `form->curField`. In this case, that is one of the strings `done`, `cancel`, `view_file`, or `file_list`. The value of the field is in `form->curValue`. This has a value for the `view_file` and `file_list` fields.

The actions in the callback `_formAction` are

"done"	The user selected the <i>Done</i> button. Closes the form, clears input using <code>axlCancelEnterFun</code> , and calls the <code>_extract</code> function to execute the data extract.
"cancel"	The user selected the <i>Cancel</i> button. Closes the form, clears input using <code>axlCancelEnterFun</code> , and calls the <code>_extract</code> function to execute the data extract.
"view_file"	The user selected the <i>View File</i> field, possibly typed an entry, and pressed <i>Return</i> . Sets the <code>view_file</code> name to the current

value of the *View File* field, letting the user type in a name.
Name must be a name on the list displayed.

"file_list"

The user picked a name from the displayed list of view file names. Name picked is *form->curValue*, and the program sets *selectedFile* (the name of the currently selected extract file) to the new value, and displays it in the *View File* field.

The *Print* and *Script* buttons have pop-ups that call predefined PCB Editor functions.

AXL Forms: Example 2

The form file *popup.form* for this is shown:

```
FILE_TYPE=FORM_DEFN VERSION=2
FORM
FIXED
PORT 50 5
HEADER "Popup Selector"
POPUP <PRINTP>
  "to File""0","to Printer""1","to Script""2".
POPUP <SCRIPTP>
  "Record""record","Replay""replay","Stop""stop".
POPUP <MYPOPUP>
  "MyPopup1" "myPopup1", "MyPopup2" "myPopup2".
TITLE
TEXT "My Popup Here:"
TLOC 1 1
ENDTEXT
FIELD my_popup
FLOC 12 3
ENUMSET 24
POP "MYPOPUP"
ENDFIELD
FIELD change_pop
FLOC 5 6
MENUBUTTON "Change" 8 3
ENDFIELD
FIELD done
FLOC 15 6
MENUBUTTON "Done" 9 3
ENDFIELD
FIELD print
FLOC 25 6
MENUBUTTON "Print" 9 3
POP "PRINTP"
ENDFIELD
FIELD script
FLOC 35 6
MENUBUTTON "Script" 11 3
POP "SCRIPTP"
ENDFIELD
ENDTILE
ENDFORM
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

Uses a form file (expected to be in the current directory) to create a pop-up. The sample program also displays in the pop-up field the value returned whenever the user selects a pop-up.

The form field `my_popup` originally has the popup values specified by the file `popup.form` (`MyPopup1` and `MyPopup2`). The AXL program responds to the *Change* button by building the pop-up display and returning the values.

```
list( list( "MyPop 1" "myPopValue1")
      list( "MyPop 2" "myPopValue2"))
```

A list of lists of display and dispatch string pairs.

```
list( list( "MyPop 12" 12) list( "MyPop 5" 5))
```

A list of lists of display and dispatch pairs, where the display value is a string, and the dispatch value is an integer.

```
list( "MyPopValue1" "MyPopValue2")
```

A list of strings, which means that each string represents both the display and dispatch values of that popup selection.

```
; formpop.il - Create and display a form with a popup
; Form call back function to respond to user selection of any field in the form
(defun _popAction (form)
  (case form->curField
    ("done"
     (axlFormClose form)
     (axlCancelEnterFun)
     t)
    ("change_pop"
     (case already_changed
       (0;Use display/dispatch string pairs
        axlFormBuildPopup(form "my_popup"
                          list(
                            list("NewPopup A" "mynewpopup_a")
                            list("NewPopup B" "mynewpopup_b"))))
        axlFormSetField(form "my_popup"
                       "My First Popups")
       )
       (1;Display string/dispatch integer pairs
        axlFormBuildPopup(form "my_popup"
                          list( list("NewPopup 12" 12)
                               list("NewPopup 5" 5)))
        axlFormSetField(form "my_popup"
                       "My Second Popups")
       )
       (t;String is both display and dispatch
        axlFormBuildPopup(form "my_popup"
                          list( "MyPopNValue1"
                                "MyPopNValue2"))
        axlFormSetField(form "my_popup"
                       "My Third Popups")
       )
    ))
```

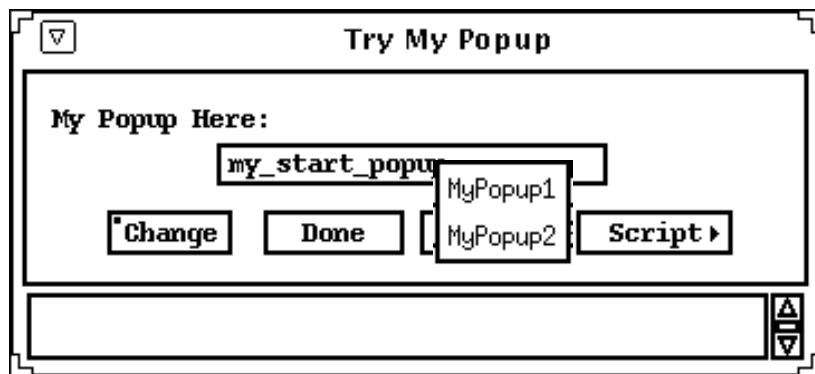
Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
)  
    already_changed++  
    t)  
  ("my_popup"  
    printf( "Got my_popup event:  
            form->curValue %s", form->curValue)  
    if( form->curValue  
        (progn  
          axlFormSetField( form "my_popup"  
                          form->curValue)))  
    t)  
  ); case  
)  
;  
; defun _popAction  
; User-callable function to set up and  
;   display the Extract Selector form  
(defun myPop ()  
  form = axlFormCreate( (gensym) "popup.form"  
                      ('("E" "OUTER") '_popAction t)  
  if( axlIsFormType(form)  
      then (print "Created form successfully.")  
      else (print "Error! Could not create form."))  
  axlFormTitle( form "Try My Popup")  
  mypopvalue = "my_start_popup"  
  axlFormSetField( form "my_popup" mypopvalue)  
  axlFormDisplay( form)  
  already_changed = 0  
); defun myPop
```

Sets the field *my_popup* to the value selected by the user and prints it.

1. Enter `myPop()` on the SKILL command line to display the **Try My Popup** form.
2. Press the middle mouse button over the pop-up field to display the original pop-up specified by the file `popup.form`.

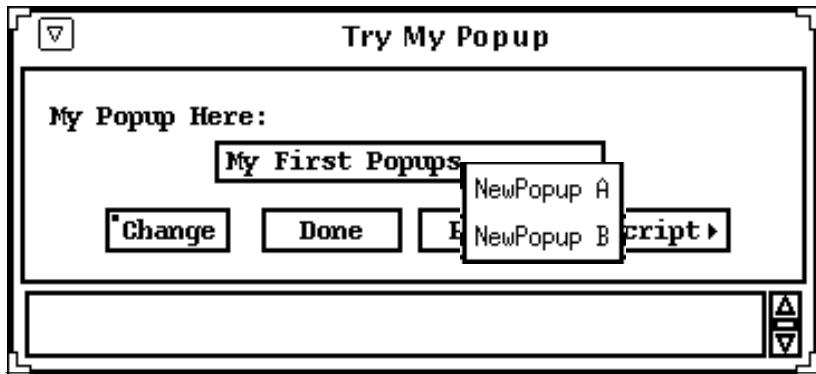


3. Click *Change*.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

The form displays the first set of pop-up values set by the program. The first pop-up values also display when you press the middle mouse button over the field.

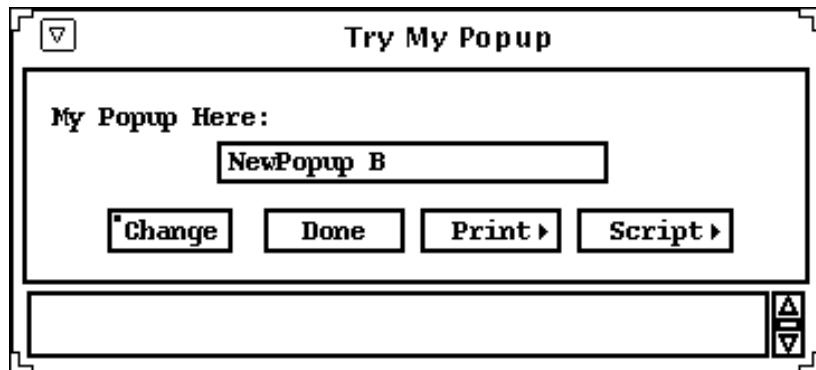


4. Make a selection.

If, for example, you selected *NewPopup B*, the program prints the following on the SKILL command line:

```
Got my_popup event: form->curValue mynewpopup_b
```

The following form is displayed.

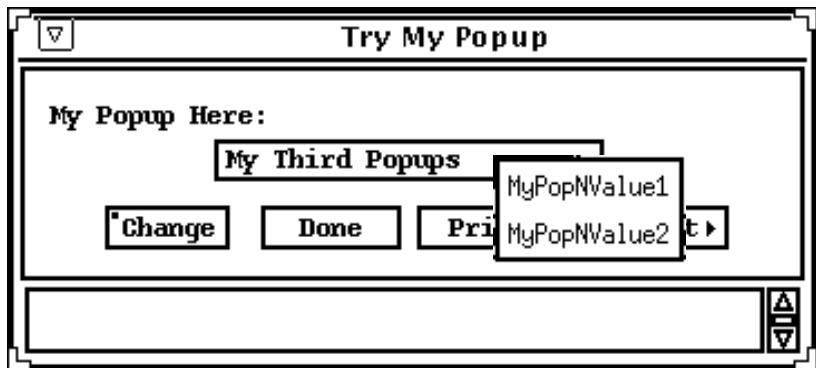


5. Click *Change*.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

The program displays the third set of pop-ups.



AXL-SKILL Form Interface Functions

This section lists the form interface functions.

axlFormBNFDoc

This is the BNF grammer for the Forms Specification Language. New options and field types are added every release. Form files are always upwards compatible but may NOT be backwards compatible if you take advantage of a new feature. Thus, a form file created in 12.0 Allegro works in 13.0 Allegro. However, if you take advantage of the TAB control (13.0) or the RIGHT justification of TEXT (13.5), you will have a form file that will not function with 12.0 of Allegro.

The following outlines the conventions used in the grammar:

[]	Optional
{}	May repeat one or more times.
<>	Supplied by user.
	Choose one or the other.
:	Definition of a token.
CAPS	Items in caps are keywords (note form parser is case insensitive)
(#)	Note: See number at end of this documentation.

BNF

form

```
FILE_TYPE=FORM_DEFN VERSION=2 (1)
FORM [form_options] (3)
formtype
PORT w h
HEADER "text"
form_header
{tile_def}
ENDFORM
```

formtype FIXED / VARIABLE

- FIXED forms have a one unlabeled TILE stanza
- VARIABLE forms have one or more label TILE stanzas
- Skill only supports FIXED form types.

PORt

- width and height of form. Height is ignored for fixed forms which auto-calculates required height. Width must be in character units.

HEADER

- initial string used in title bar of form (may be overridden by application).

form_header

```
[{default_button_def}]  
[{popup_def}]  
[{message_def}]
```

default_button_def

DEFAULT <label>

- sets the default button to be <label>. If not present form sets default button to one of the following:
ok (done), close, cancel.
- label must be of type MENU_BUTTON.

popup_def

POPUP <>popupLabel> {"<display>", "<dispatch>"}.

- popups may be continued over several lines by using the backslash () as the last character on line.

message_def

MESSAGE messageLabel messagePriority "text".

form_options

[TOOLWINDOW]

- this makes a form to be a toolwindow which is a floating toolbar. It is typically used as a narrow temp window to display readouts.

[FIXED_FONT]

- by default forms use a variable width font, this sets this form to use a fixed font. Allegro/Apd use mostly variable width while Allegro PCB SI and SigXplorer use fixed width fonts.

[AUTOGREYTEXT]

- when a fillin or enum control is greyed, grey static text to the left of it.

[UNIXHGT]

- works around a problem with Mainsoft in 15.0 where a button is sandwiched vertically between 2 combo/fillin controls. The button then overlaps these controls. This adds extra line-2-line spacing to avoid this. You should only use this option as a last resort. In a future release it may be treated as a Nop. On Windows this is ignored.

tile_def

```
TILE [<titleLabel>]      (4)
[TPANEL tileType]
{{text_def}}
{{group_def}}
{{list_def}}
{{field_def}}
{{button_def}}
{{grid_def}}
{{flex_def}}
ENDTILE
```

tabset_def

```
TABSET [label]
[OPTIONS tabsetOptions]
FLOC x y
FSIZE w h
{tab_def}
ENDTABSET
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

tab_def

```
TAB "<display>" [<label>]      (10)
[{{text_def}}]
[{{group_def}}]
[{{field_def}}]
[{{grid_def}}]
ENDTAB
```

text_def

```
TEXT "display" [label] (9)
FLOC x y
[FSIZE w h]      (8)
text_type
[OPTIONS textOptions]
ENDTEXT
```

text_type

```
[INFO label w] |
[THUMBNAIL [<bitmapFile>|#<resource>] ]
```

group_def

```
GROUP "display" [label] (9)
FLOC x y
FSIZE w h      (8)
[INFO label]
ENDGROUP
```

list_def

```
FIELD label
FLOC x y
LIST "" w h
list_options
ENDFIELD
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

field_def

```
FIELD label
FLOC x y
[FSIZE w h]      (8)
field_type
field_options
ENDFIELD
```

button_def

```
FIELD label
FLOC x y
[FSIZE w h]      (8)
MENUBUTTON "display" w h
button_options
ENDFIELD
```

grid_def

```
GRID fieldName
FLOC x y
FSIZE w h      (8)
[OPTIONS INFO | HLINES | VLINES | USERSIZE ]
[POP "<popupName>"]

[GHEAD TOP|SIDE]
[HEADSIZE h|w]
[OPTION 3D|NUMBER]
[POP "<popupName>"]
[ENDGREADH]
ENDGRID
```

field_type

```
REALFILLIN w fieldLength |
LONGFILLIN w fieldLength |
STRFILLIN w fieldLength |
INTSLIDE BAR w fieldLength |
ENUMSET w [h] |          (11)
CHECKLIST "display" ["radioLabel"] |
LIST "" w h |
```

```
TREEVIEW w h |
COLOR w h |
THUMBNAIL [<bitmapFile>|#<resource>] |
PROGRESS w h
TRACKBAR w h
```

field_options

[INFO_ONLY]

- sets field to be read only.

[POP "<popupName>"]

- assigns a popup with the field.
- a POPUP definition by the same name should exist.
- supported by field_types: xxxFILLIN, INTSLIDE BAR, MENUBUTTON, ENUMSET

[MIN <value>]

[MAX <value>]

- assigns a min and/or max value that field might have.
- both supported by field_types: LONGFILLIN, INTSLIDE BAR, REALFILLIN.
- value by either an integer or floating point number.

[DECIMAL <accuracy>]

- assigns a floating min and/or max value that field might have.
- assigns number of decimal places field has (default is 2)
- both supported by field_types: REALFILLIN

[VALUE "<display>"]

- initial field value.
- supported by field_types: xxxFILLIN

[SORT]

- alphanumerical sorted list (default order of creation)
- supported by field_type: LIST

[OPTIONS dispatchsame]

- for enumset fields only.
- if present will dispatch to application drop-down selection even if the same as current. By default, the form's package filters out any user selection if it is the same as what is currently displayed.

[OPTIONS prettyprint]

- for enumset fields only
- displays contents of ENUM field in a visually pleasing way

[OPTIONS ownerdrawn]

- for enumset fields only
- used to display color swatches in an ENUM field. See `axlFormBuildPopup`.

list_options

[OPTIONS sort|alphanumsort|prettyprint|multiselect]

sort - conversion alphabetical sort
alphanumsort - sort so NET10 appears after

NET2 prettyprint - make more readable, convert case.

All dispatch entries will be upper case multiselect - multi-select list box.
User can select more than one item (follows Microsoft selection model).

x

y

w

h

- display geometry (integers).

- all field, group and text locations are relative to the start of the tile they belong or to the start of the form in the case of FIXED forms.

- x & h are in CHARHEIGHT/2 units.

- y & w are in CHARWIDTH units.

button_options

[MULTILINE]

- wraps button text to multiple lines if text string is too long for a single line.

dispatch

- string that is dispatched to the code.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

display

- string that is shown to user

bitmapFile

- name of a bmp file. Assumes can be found via BITMAPPATH

resource

- integer resource id (bitmap must be bound in executable via the resource file). '#' indicates it is a resource id.
- not support in AXL forms.

fieldLength

- maximum width of field. Field will scroll if larger then field display width.

label

- named used to access field from code. All fields should have unique names.
- should use lower case.

messageLabel

- name used to allow code to refer to messages.
- case insensitive

messagePriority

- message priority 0 - info (not in journal file), 1 - info, 2 - warning, 3 - error, 4 fatal (display in message box).

radioLabel

- named used to associate several CHECKLIST fields as a radio button set. All check fields should be given the same radioLabel.
- should use lower case.

textOptions

[RIGHT | CENTER | BORDER | BOLD | UNDERLINE]

- TEXT/INFO field type.
- text justification, default is left.
- BORDER: draw border around text.

[STRETCH]

- THUMBNAIL field type.
- stretch bitmap to fit thumbnail rectangle, default is center bitmap.

tabsetOptions

tabsetDispatch]

- By default tabs dispatch individual tabs as separate events. This is not always convenient for certain programming styles. This changes the dispatch mode to be upon the tabset where a selection of a tab causes the event field=tabsetLabel value=tabLabel.

The default is:

field=tabLabel value=t

Script record/replay remains based upon tab in either mode.

tileLabel

- name used to allow code to refer to this tile.
- should use lower case.
- only applies to VARIABLE forms.
- not support with AXL forms.

tileType [0/1/2]

- 0 top tile, 1 scroll tile, 2 bottom tile.
- only applies to VARIABLE FORMS.
- region where tile will be instantiated. Forms have 3 regions top, bottom and scroll (middle).
- not support with AXL forms.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

flex_def - rule based control sizing upon form resize (see axlFormFlex)

[FLEXMODE <autorule>
[FLEX <label> fx fy fw fh]

FLEXMODE <autoRule>

FLEX fx fy fw fz

- see axlFormFlexDoc

autorule - generic sizing placement rule

fx
fy
fw
fh

- floating value between 0 and 1.0

Note:

- ❑ FILE_TYPE line must always appear as the first line of form file in format shown.
- ❑ Form files must have a .form extension.
- ❑ There may only be one FORM in a form file.
- ❑ There must be one and only one TILE defintion in a FIXED form file. <tileLabel> and TPANEL are not required.
- ❑ Unless otherwise noted limits are as follows:
labels - 128
title - 1024
display - 128 except for xxxFILLIN types which are 1024
- ❑ Additional items may appear in existing form files (FGROUP) but they are obsolete and are ignored by the form parser. REALMIN & REALMAX are obsolete and replaced by MIN and MAX respectively. They will still be supported and are mapped to MIN and MAX.
- ❑ For grid_def two headers (side and top) are maximum.
SIZE - most controls determine the size from the text string. You are required to provide FSIZE for GROUP, GRID, TREEVIEW and LIST controls. For TEXT controls if FSIZE is provided after it overrides the width calculated by the text length and if present the INFO width. If the INFO line appears you should put the FSIZE line after it.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

- ❑ Both TEXT and GROUP support optional label on their definition line. This was added as a convenience in supporting FLEX capability. If application wishes to dynamically modify the text the INFO keyword is normally used. When both are present the INFO keywords takes precedence.
 - ❑ If the optional label for TABS is not provided, the field display name is used. Any spaces within the field display name are replaced by underscores ("_").
 - ❑ The height ([h]) for ENUMSET is optional. When not set (the default) the drop-down is only presented under user control. When height greater than 1 then the dropdown is always visible (Microsoft SIMPLE drop-down). You only want to use this feature in forms that can afford the space consumed by the drop-down.
-

The forming syntaxes are NOT supported by the formeditor.

This following syntax is supported and may be placed anywhere in the form file to support conditional processing of the form file:

```
#ifdef <variable>
{ }

{ #elseif <variable>
  }

{ #else
  }
}
```

axlFormCallback

```
formCallback(  
    [r_form]  
)  
==> t
```

Description

This is not a function but documents the callback interface for form interaction between a user and Skill code. The Skill program author provides this function.

When the user changes a field in a form the Allegro form processor calls the procedure you specified as the `g_formAction` argument in `axlFormCreate` when you created that form. The form attribute `curField` specifies the name of the field that changed. The form attribute `curValue` specifies the current value of the field (after the user changed it). If you set `g_stringOption` to `t` in your call to `axlFormCreate` when you created that form, then `curValue` is a string. If `g_stringOption` was `nil` (the default), then `curValue` is the type you specified for that field in the form file.

Note: The term `formCallback` used in the title of this callback procedure description is a dummy name. The callback function name must match the name or symbol name you used as the `g_formAction` argument in `axlFormCreate` when you created the form.

If you specify the callback name (`g_formAction`) as a string in your call to `axlFormCreate`, SKILL calls that function with no arguments. If you specify `g_formAction` as a symbol, then SKILL calls that function with the form handle as its single argument.

The callback must call `axlFormClose` to close the form and to continue in the main application code if form mode is blocking.

All form information is provided by the `r_form` argument which is a form data type. Applications can extend the data stored on this type by adding their own attributes. Please capitalize the first letter of the attribute name to avoid conflicts with future additions by Cadence to this structure. Tables 1 and 2 show the available field types and how they impact the `r_form` data type.

Table 1

Form Field Types:

Type	What the field is commonly known to the user.
Keyword	How the field is declared in the form file (see <code>axlFormBNFDoc</code>).
curValue	The data type seen in the form dispatch and <code>axlFormGetField</code> (see <code>axlFormCallback</code>).
curValueInt	If <code>curValue</code> can be mapped to an integer. For certain field types provides additional information.

Type	Keyword	cuValue	curValueInt
Button	MENUBUTTON (6)	t	1
Check Box	CHECKLIST (1)	t / nil	1 or 0
Radio Box	CHECKLIST (1)	t / nil	1 or 0
Long (integer)	INTFILLIN	integer	integer
Real (float)	REALFILLIN	floating point	N/A
String	STRFILLIN	string	N/A
Enum (popup)	ENUMSET	string	integer (2)
List	LIST	string	index
Color well	COLOR	t / nil	1 or 0
Tab	TABSET/TAB	string or t (3)	N/A or 1/0
Tree	TREEVIEW	string	See: <code>axlFormTreeViewSet</code>
Text	INFO (4)	N/A	N/A
Graphics	THUMBNAIL (5)	N/A	N/A
Trackbar	TRACKBAR	integer	integer
Grid	GRID		See: <code>axlFormGridDoc</code>

Note:

- ❑ What distinguishes between a radio button and check box is that a radios buttons are a group of check boxes where only one can be set. To related several checkboxes as radio buttons use supply the same label name as the third field (`groupLabel`) in the form file description:

```
CHECKLIST <fieldLabel> <groupLabel>
```

When a user sets a radio button the button be unset will dispatch to the app's callback with a value of `nil`.

- ❑ `Enum` will only set `curValueInt` on dispatch when the dispatch value of their popup uses an integer. Otherwise this field is `nil`.
- ❑ Tabs can dispatch in two methods:
 - default when a tab is selected your dispatcher receives the tab name in the `curField` and `curValue` is `t`.
 - If `OPTIONS tabsetDispatch` is set in the `TABSET` of the form file then when a tab is selected your app dispatser receives the `TABSET` as the `curField` and the `curValue` being the name of the TAB that was selected.
- ❑ `INFO` fields can be static where the text is declared in the form file or dynamic where you can set the text via the application at run-time. To achieve dynamic access enter the following in the form file:

```
TEXT "<optional initial text>"
```

```
INFO <fieldLabel>
```

... reset of `TEXT` section ...

- ❑ Thumbnails support three methods:
 - Static bitmap declared via form file.
 - Bitmaps that can by changed by the application at run-time.
 - Basic drawing canvas (see `axlGRPDoc`).
- ❑ Buttons are stateless. The application cannot set the button to the depressed state. You can only use `axlFormSetField` to change the text in the button. Several button fieldLabels are reserved. Use them only as described:

Done or *Ok*

Do action and close form.

Cancel

Cancel changes and close form.

Print

Print form; do not use.

Help

Call `cdsdoc` for help about form. Do not use.

Table 2

Attribute Name	Set?	Type*	Description
curField	no	string	Name of form field (control) that just changed.
curValue	no	See->	Value dependant upon field type (2).
curValueInt	no	See->	Value dependant upon field type (2).
doneState	no	int	0 = action; 1 = done; 2 = cancel; 3 = abort (1)
form	no	string	Name of this form (form file name).
isChanged	no	t / nil	t = user has changed one or more fields.
isValueString	no	t / nil	t all field values are strings. nil one or more fields are not strings.
objType	no	string	Type of object; in this case form.
type	no	string	Always fixed.
fields	no	list of strings	All fields in the form (3).
infos	no	list of strings	All info. fields in form (3).
event	no	symbol	List, tree and grid control only. See <code>axlFormGridDoc</code> for grid info.
row	no	integer	Grid control only.
col	no	integer	Grid control only.
treeViewSelState	no	integer	Tree control only.

Note:

- ❑ The `doneState` shows 0 for most actions. If a button with *Done* or *Ok* is pushed, then the done state is set. A button with the *Cancel* label sets the cancel state. In either the Done or Cancel state, you need to close the form with `axlFormClose`. If the abort state is set, the form closes even if you do not issue an `axlFormClose`.

- ❑ Data type is dependant upon the field type, see Table 1.
- ❑ The difference between the fields and infos list is that items appearing in the infos list are static text strings the program can change at run-time. All other labels appear in the fields list and are reflect they can be changed by the user (even buttons, tabs, greyed and hidden fields).
- ❑ Event for list box is `t` if item is selected, `nil` if deselected. This is always `t` for single select list box while the multi-select option can have both states.
- ❑ Event and `treeViewSelState` for a tree control see `axlFormTreeViewAddItem`.

Arguments

r_form Form dbid.

Value Returned

`t` Always returns `t`.

Examples

See `axlFormCreate` and `axlFormBuildPopup` examples.

axlFormCreate

```
axlFormCreate(  
    s_formHandle  
    t_formfile  
    [lt_placement]  
    g_formAction  
    g_nonBlock  
    [g_stringOption]  
)  
⇒ r_form/nil
```

Description

Creates a form structure based on the form descriptive file *t_formfile*. This call only supports forms of type “fixed” and will fail if *t_formfile* contains any variable tiles. This function does not display the form. Use `axlFormDisplay` to display a form.

Note: If *s_formHandle* is an existing *r_form*, then `axlFormCreate` does not create a new form, but simply exposes and displays the existing form, *s_formHandle*, and returns `nil`.

Arguments

<i>s_formHandle</i>	Global SKILL symbol used to reference form. Note: Do not use the same symbol to reference different form instances.
<i>t_formfile</i>	Filename of the form file to be used to define this form. <code>axlFormCreate</code> uses the PCB Editor environment variable, <code>FORMPATH</code> , to find the file, if <i>t_formfile</i> is not a full path. The filename, by convention, should use the <code>.form</code> extension.
<i>lt_placement</i>	Form placement. PCB Editor uses its default placement if this argument is <code>nil</code> . See Window Placement on page 391

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

g_formAction Specifies the SKILL commands (callbacks) to execute after every field change (Note that this is very different from Opus forms). You can set this to one of the formats shown:

g_formAction Options

Option	Description
<i>t_callback</i>	String representation of the SKILL command to be executed.
<i>s_callback</i>	Symbol of the SKILL function to be called (passes the <i>r_form</i> returned from <code>axlFormCreate</code> as its only parameter.)
<i>nil</i>	<code>axlFormDisplay</code> blocks until the user closes the form. You must place a <i>Done</i> button (field name <code>done</code>) and optionally a <i>Cancel</i> button (field name <code>cancel</code>) in the form for <i>g_formAction</i> to function properly. The user can access all of the fields and values using the <i>r_form</i> user type.

g_nonBlock If *g_nonBlock* is *t*, the form runs in non-blocking mode. In blocking mode (the default), `axlFormDisplay` blocks until the user closes the form. Blocking is an easier programming mode but might not be appropriate for your application. If the callback (*g_formAction*) is *nil*, then `axlFormDisplay` ignores *g_nonBlock*, and the form runs in blocking mode.

Use of blocking mode blocks the progress of the SKILL code, but does not prevent other PCB Editor events from occurring. For example, if blocked, users can start the *Add Line* command from PCB Editor menus.

g_stringOption If *t*, the form returns and accepts all values as strings. By default, it returns and accepts values in the format declared in the form file.

Value Returned

r_form *dbid* of form created.

nil No form created.

Example

See [AXL Forms: Example 1](#) on page 477.

axlFormClose

```
axlFormClose(  
    r_form  
)  
⇒ t/nil
```

Description

Closes the form *r_form*. Unless the form is running without a callback handler, you must make this call to close the form. Without a registered dispatch handler, PCB Editor closes the form automatically before returning to the application from `axlFormDisplay`.

Note: `axlUIWClose` also performs the same function.

Arguments

r_form Form *dbid*.

Value Returned

t Closed the form.

nil Form was already closed.

Example

See [AXL Forms: Example 1 on page 477](#):

```
(case form->curField  
  ("done"  
   (axlFormClose form)  
   (axlCancelEnterFun)  
   (_extract)  
   t))
```

axlFormDisplay

```
axlFormDisplay(  
    r_form  
)  
⇒ t/nil
```

Description

Displays the form *r_form* already created by `axlFormCreate`. For superior display appearance, set all the field values of the form before calling this function. A form in blocking mode blocks until the user closes the form.

If a form is already displayed, this function simply exposes it.

Arguments

r_form Form *abid*.

Value Returned

t Successfully opened or exposed the form.

nil Failed to open or expose the form.

Example

See [AXL Forms: Example 1](#) on page 477.

```
axlFormDisplay( form)
```

axlFormBuildPopup

```
axlFormBuildPopup(  
    r_form  
    t_field  
    l_pairs  
)  
⇒ t/nil
```

Description

Builds a pop-up for field *t_field* in the open form *r_form*. Field must be an enumerated list type field. `axlFormBuildPopup` replaces the existing pop-up attached to the field. The *l_pairs* argument is a list of display and dispatch string pairs.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of form field.
<i>l_pairs</i>	Must be one of the formats described. Each list object defines a single popup entry. Options can be 1 or 2 additional list options that are <i>S_color</i> / <i>x_color</i> for enum field types with color; bold or underline for bold or underlined items.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

I_pairs Format Options

Option	Description	Example
List of lists of string pairs	The first member of each string pair list is the display value—the string displayed in the pop-up. The second member of each string pair is the dispatch value—the string value returned as <i>form->curValue</i> when the user selects that pop-up entry.	(list (list "MyPop A" "myvalue_a") list("MyPop B" "myvalue_b"))
List of lists of pairs	List of lists of pairs where the first member of each pair is a string giving the display value, and the second member is an integer that is the dispatch value, returned as <i>form - curValue</i> when the user selects that pop-up entry. You can use the return value as an index into an array.	(list (list "MyPop A" 5) list("MyPop B" 7))
List of strings	Uses each string both for display value and the return value.	(list "MyPop A" "MyPop B")

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

Option	Description	Example
Optional field	<p>Specifies a color swatch. This is currently only supported by ENUM field types (it is ignored by other field types). With an ENUM you need to add OPTIONS ownerdrawn in the form file for the FIELD in question to see the color swatch in the popup. You can use either pre-defined color names (see axlColorDoc) or Allegro board colors (see axlLayerGet).</p>	<p>You can't mix this color type in a single popup.</p> <pre>'((("Green" 1 green) ("Red" 2 red) ("Yellow" 3 yellow)) ('(("Top" "top" 2) ("Gnd" "gnd" 4) ("Bottom" "btm" 18)))</pre> <p>If instead of a color or Allegro color number, you provide a nil, then that popup entry will not have a color swatch.</p> <pre>'((("None" 0 nil) ("Green" 1 green) ("Red" 2 red) ("Yellow" 3 yellow))</pre> <p>Font type of bold or underline can be specified via:</p> <pre>'((("Top" "top" bold) ("Gnd" "gnd" underline) ("Bottom" "btm")))</pre> <p>When font type is combined with color it looks like:</p> <pre>'((("Top" "top" "Green" bold) ("Gnd" "gnd" "Red" underline))</pre>

Notes:

- Allows a maximum of 256 pop-up entries in one pop-up.
- All entries in an *l_pairs* argument must be the same type of format. That is, you cannot have a list containing both display/dispatch strings and display/enum types, or display/dispatch and single-string entries.

Value Returned

t Field set.

nil Field not set.

Example

See [AXL Forms: Example 2](#) on page 481.

axlFormGetField

```
axlFormGetField(  
    r_form  
    t_field  
)  
⇒ g_value/nil
```

Description

Gets the value of *t_field* in the open form *r_form*. The value is a string if *g_stringOption* was set in `axlFormCreate`. Otherwise the value is in the field type declared in the form file.

Arguments

r_form Form *dbid*.

t_field Name of field.

Value Returned

g_value Current value of the field.

nil Field does not exist, or false if boolean field such as check box or radio button.

Example

1. Load the example code given in [AXL Forms: Example 1](#) on page 477.
2. Enter the command `myExtract()` on the SKILL command line.

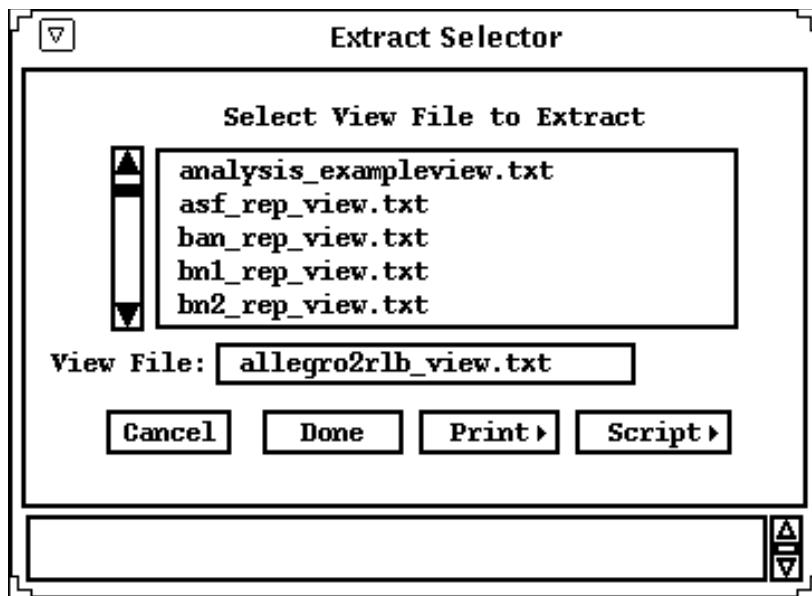
The command displays the **Extract Selector** form, listing all available extract view files.

3. Select any file in the list, or type a name into the *View File* field.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

allegro2rlb_view.txt is entered.



```
axlFormGetField( form "view_file")
⇒ "allegro2rlb_view.txt"
```

Examines the value of "*view_file*".

axlFormListDeleteAll

```
axlFormListDeleteAll(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Deletes all the items from the form list field, *t_field*.. Use `axlFormListDeleteAll` to clear an entire list field to update it using `axlFormSetField`, then display it using `axlFormSetField` on the field with a `nil` field value.

Arguments

r_form Form *dbid*.

t_field Name of field.

Value Returned

t All items deleted properly.

nil All items not deleted.

Examples

In this example you do the following:

1. Use the `axlFormCreate` examples to create and display the Extract Selector dialog box shown in [Figure 11-1](#) on page 514.
2. On the SKILL command line, enter:

```
axlFormListDeleteAll(form "file_list")  
==> nil
```

The list is removed from the dialog box as shown in [Figure 11-2](#) on page 514.

3. On the SKILL command line, enter:

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
axlFormSetField(form "file_list" "fu")
axlFormSetField(form "file_list" "bar")
axlFormSetField(form "file_list" nil)
==> t
```

The Extract Selector dialog box is displayed with new list as shown in [Figure 11-3](#) on page 515.

Figure 11-1 Extract Selector Dialog Box

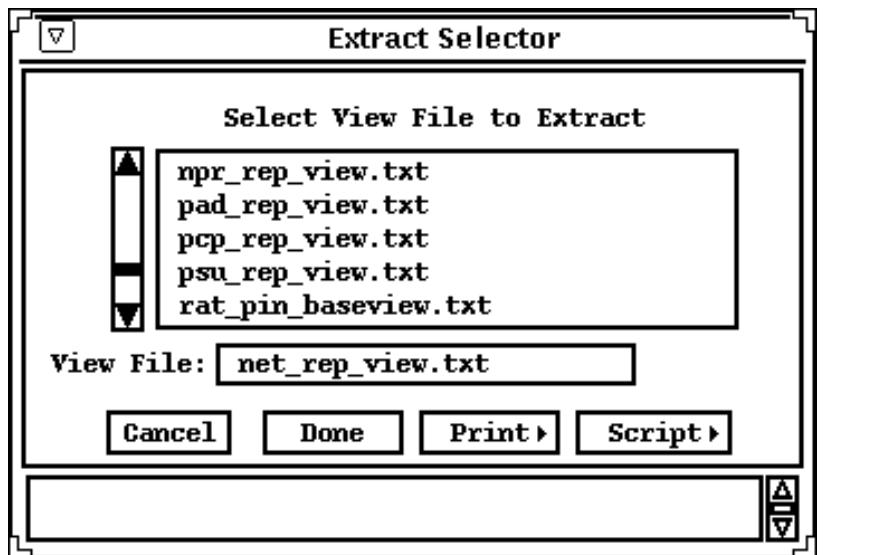
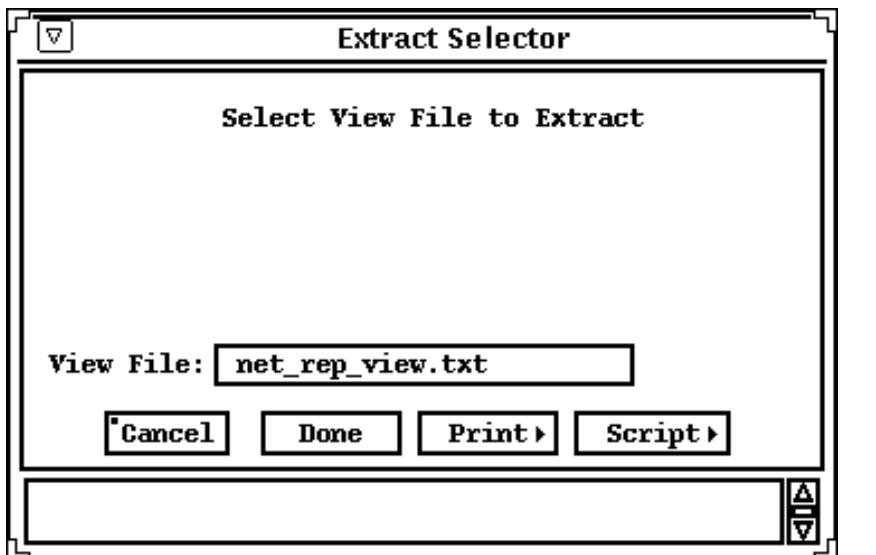
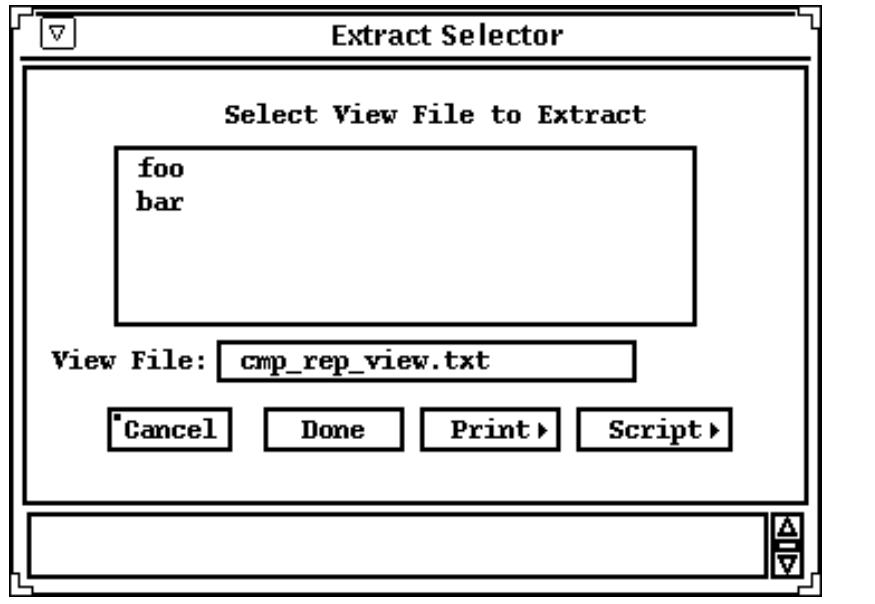


Figure 11-2 Extract Selector Dialog Box - List removed



Allegro PCB and Package User Guide: SKILL Reference
Form Interface Functions

Figure 11-3 The Extract Selector dialog box - Displayed with a new list



axlFormListSelect

```
axlFormListSelect(  
    r_form  
    t_field  
    t_listItem/nil  
)  
⇒ t/nil
```

Description

Highlights, and if not visible in the list, shows the designated item. Since PCB Editor forms permit only one item to be visible, it deselects any previously selected item. If *nil* is passed for *t_listItem* the list is reset to top and the selected list item is deselected.

Arguments

<i>r_form</i>	Form id
<i>t_field</i>	Name of field.
<i>t_listItem/nil</i>	String of item in the list. Send <i>nil</i> to deselect any selected item and set list back to top.

Value Returned

<i>t</i>	Highlights item. Arguments are valid.
<i>nil</i>	Arguments are invalid.

axlFormSetField

```
axlFormSetField(  
    r_form  
    t_field  
    g_value  
)  
⇒ t/nil
```

Description

Sets *t_field* to value *g_value* in open form *r_form*. Must pass the correct type, matching the entry in the form value or string type.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of field.
<i>g_value</i>	Desired value of field.

Value Returned

<i>t</i>	Field set to desired value.
<i>nil</i>	Field not set to the desired value due to invalid arguments.

Example

See [AXL Forms: Example 1](#) on page 477.

```
axlFormSetField( form "file_list" fileName)
```

axlFormSetInfo

```
axlFormSetInfo(  
    r_form  
    t_field  
    t_value  
)  
⇒ t/nil
```

Description

Sets info *t_field* to value *t_value* in open form *r_form*. Unlike `axlFormSet`, user cannot change an info field.

Note: You can also use `axlFormSetField` for this function.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of field.
<i>t_value</i>	Desired value of field.

Value Returned

<i>t</i>	Field was set to desired value.
<i>nil</i>	Field not set to desired value due to invalid arguments.

Example

See the use of `axlFormSetField` in the [“AXL Forms: Example 1”](#) on page 477.

```
axlFormSetInfo( form "file_list" fileName)
```

axlFormTest

```
axlFormTest(  
    t_formName  
) r_form/nil
```

Description

This is a development function for test purposes. Given a form file name this opens a form file to check for placement of controls. If form uses standard button names (for example, *ok*, *done*, *close*, *cancel*), you can close it by clicking the button. Otherwise, use the window control. If form is currently open, exposes form and returns.

Arguments

t_formName Name of form.

Value Returned

Form handle if successfully opens.

Example

Open PCB Editor drawing parameter form:

```
axlFormTest( "status" )
```

axlFormRestoreField

```
axlFormRestoreField(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Restores the *t_field* in the open form *r_form* to its previous value. The previous value is only from the last user change and not from the form set field functions. This is only useful in the *form callback* function.

Use in the *form callback* to restore the previous value when you detect the user has entered an illegal value in the field.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of field.

Value Returned

<i>t</i>	Field restored.
<i>nil</i>	Field not restored and may not exist.

Example

See “[AXL Forms: Example 1](#)” on page 477 where the callback function checks that the user has entered a filename that is on the list of available extract view filenames. If the user-entered value is not on the list, then the program calls `axlFormRestoreField` to restore the field to its previous value.

```
(case form->curField
  ("view_file"
    (if form->curValue
      (progn
        ; Accept user input only if on list
        if(member( form->curValue fileList)
          then axlFormSetField( form
            "view_file" form->curValue)
          else axlFormRestoreField(
            form "view_file")))))
  t)
```

axlFormTitle

```
axlFormTitle(  
    r_form  
    t_title  
)  
⇒ t/nil
```

Description

Overrides title of the form.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_title</i>	String to be used for new form title

Value Returned

t	Changed form title.
nil	No form title changed.

Example

See “[AXL Forms: Example 1](#)” on page 477.

```
axlFormTitle( form "Extract Selector")
```

axlIsFormType

```
axlIsFormType(  
    g_form  
)  
⇒ t/nil
```

Description

Tests if argument *g_form* is a form *dbid*.

Arguments

g_form *dbid* of object to test.

Value Returned

t *r_form* is the *dbid* of a form.

nil *r_form* is not the *dbid* of a form.

Example

```
form = axlFormCreate( (gensym)  
    "extract_selector.form" ' ( "E" "OUTER" )  
    '_formAction t)  
if( axlIsFormType(form)  
    then (print "Created form successfully.")  
    else (print "Error! Could not create form."))
```

Checks that the form you create is truly a form.

axlFormSetFieldVisible

```
axlFormSetFieldVisible(  
    r_form  
    t_field  
    x_value  
)  
⇒ t/nil
```

Description

Sets a form field to visible or invisible.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Form field name (string).
<i>x_value</i>	1 - set field visible or 0 - Set field invisible

Value Returned

t	Form field set visible.
nil	Form field set invisible.

axlFormIsFieldVisible

```
axlFormIsFieldVisible(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Determines whether a form field is visible.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Form field name (string).

Value Returned

t	Form field is visible.
nil	Form field is not visible.

Callback Procedure: formCallback

```
formCallback(  
    [r_form]  
)  
⇒ t
```

Description

This is not a function but documents the callback interface for form interaction between a user and SKILL code. The SKILL programmer provides this function.

When the user changes a field in a form, the PCB Editor form processor calls the procedure you specified as the *g_formAction* argument in `axlFormCreate` when you created that form. The form attribute *curField* specifies the name of the field that changed. The form attribute *curValue* specifies the current value of the field (after the user changed it). If you set *g_stringOption* to `t` in your call to `axlFormCreate` when you created that form, then *curValue* is a string. If *g_stringOption* was `nil` (the default), then *curValue* is the type you specified for that field in the form file.

Note: The term `formCallback` used in the title of this callback procedure description is a dummy name. The callback function name must match the name or symbol name you used as the *g_formAction* argument in `axlFormCreate` when you created the form.

If you specify the callback name (*g_formAction*) as a string in your call to `axlFormCreate`, SKILL calls that function with no arguments. If you specify *g_formAction* as a symbol, then SKILL calls that function with the form handle as its single argument.

The callback must call `axlFormClose` to close the form and to continue in the main application code if form mode is blocking.

All form information is provided by the *r_form* argument which is a form data type. Applications can extend the data stored on this type by adding their own attributes. Please capitalize the first letter of the attribute name to avoid conflicts with future additions by Cadence to this structure. [Table 11-3](#) on page 527 and [Table 11-4](#) on page 529 show the available field types and how they impact the *r_form* data type.

[Table 11-3](#) on page 527 describes Form Field Types using the following:

Type	What the user calls the field
Keyword	What the form file calls the field

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

curValue	Data type seen in the form dispatch and axlFormGetField. See Callback for more information.
curValueInt	Additional information for certain field types that can be mapped to integers.

Table 11-3 Form Field Types

Type	Keyword	curValue	curValueInt
Button	MENUBUTTON	dispatch action only (t)	1
Check Box	CHECKLIST	t/nil	0 or 1
Radio Button	CHECKLIST	t/nil	0 or 1
Long (integer)	INTFILLIN	integer number	Integer
Real (float)	REALFILLIN	float number	n/a
String	STRFILLIN	string	n/a
Enum (popup)	ENUMSET	string	Possible integer ¹
List	LIST	string	Offset from start of list (0 = first entry).
Color well	COLOR	t/nil	1 or 0
Tab	TABSET/TAB	string or t	n/a or 1/0
Tree	TREEVIEW	string	see axlFormTreeViewSet
Text	INFO	n/a	n/a
Graphics	THUMBNAIL	n/a	n/a
GRID	GRID	see Using Grids on page 470	

¹. Integer if the dispatch value of the pop-up is an integer.

Notes:

- What distinguishes between a radio button and a check box is that radio buttons are a group of boxes where only one can be set. To relate several checkboxes as radio buttons, supply the same label name as the third field (groupLabel) in the form file description:

```
CHECKLIST <fieldLabel> <groupLabel>
```

When a user sets a radio button, the button being unset will dispatch to the application's callback with a value of nil.

- Enum will only set curValueInt on dispatch when their dispatch value of their popup uses an integer. Otherwise this field is nil.
- Tabs can dispatch in two methods:
 - Default when a tab is selected, your dispatcher receives the tab name in the curField and curValue is t.
 - If "OPTIONS tabsetDispatch" is set in the TABSET of the form file, then when a tab is selected your application dispatcher receives the TABSET as the curField and the curValue being the name of the TAB that was selected.
- INFO fields can be static where the text is declared in the form file or dynamic where you can set the text via the application at run-time. To achieve dynamic access, enter the following in the form file:

```
TEXT "<optional initial text>"  
INFO <fieldLabel>  
... reset of TEXT section ...
```

- Thumbnails support the following methods:
 - static bitmap declared via the form file
 - bitmaps that can be changed by the application at run-time
 - basic drawing canvas -- see [Chapter 12, "Simple Graphics Drawing Functions"](#)
- Buttons are stateless. The application cannot set the button to the depressed state. You can only use axlFormSetField to change the text in the button. Several button fieldLabels are reserved. Use them only as described:

done or OK	Do action and close the form.
cancel	Cancel changes and close the form.
print	Print the form -- do not use.
help	Call cdsdoc for help about the form -- do not use.

Table 11-4 Form Attributes

Attribute Name	Set?	Type*	Description
<i>curField</i>	no	string	Name of form field just changed
<i>curValue</i>	no	See -->	Depends on value of <i>curField</i> (string, int, float, boolean)
<i>curValueInt</i>	no	See -->	Depends on value of <i>curField</i> field
<i>doneState</i>	no	int	0 = action; 1 = done; 2 = cancel; 3 = abort
<i>form</i>	no	string	Name of this form
<i>isChanged</i>	no	t/nil	t = user has changed one or more fields in form.
<i>isValueString</i>	no	t/nil	t = all field values are strings nil = one or more fields are not strings
<i>objType</i>	no	string	Type of object, in this case "form"
<i>type</i>	no	string	Form type, always "fixed"
<i>fields</i>	no	list of strings	All fields in the form.
<i>infos</i>	no	list of strings	All info fields in the form.
<i>event</i>	no	symbol	Grid control only -- see Using Grids on page 470
<i>row</i>	no	integer	Grid control only
<i>col</i>	no	integer	Grid control only
<i>treeViewSelState</i>	no	integer	Tree control only

* You can add your own attribute types to the form type. It is recommended you capitalize the first letter of the name to avoid conflict with future PCB Editor releases.

Notes:

- The *doneState* shows 0 for most actions. Selecting a *Done* or *OK* button sets the done state. Selecting a *Cancel* button sets the cancel state. With the done or cancel state set, you use *axlFormClose* to close the form. Setting the abort state closes the form, even if you do not issue an *axlFormClose* command.

- Data type is dependant on the field type. See [Table 11-3](#) on page 527 for more information on Form Field Types.
- The infos list is different from the fields list. The infos list comprises static text strings that the program can change at run-time. The fields list comprises all other labels which can be changed by the user including even those on buttons and tabs, greyed and hidden fields.

Arguments

r_form Form *dbid*.

Value Returned

t Always returns *t*.

Example

See [axlFormCreate](#) on page 502 and [axlFormBuildPopup](#) on page 507 for examples.

axlFormAutoSize

```
axlFormAutoSize(  
    r_form  
)  
⇒ t/nil
```

Description

Resizes a form to fit its controls. Recalculates the required width and height and resizes the form based on the current visibility of the form's fields.

Arguments

<i>r_form</i>	Form handle.
<i>t_field</i>	Form field name (string).

Value Returned

t	Form resized.
nil	<i>r_form</i> does not reference a valid form.

axlFormColorize

```
axlFormColorize(  
    o_form  
    t_field  
    g_option  
    g_color  
)  
⇒ t/nil
```

Description

Allows the override of background and/or text color of a control. Only the following controls are supported:

- STRFILLIN
- READFILLIN
- LONGFILLIN
- INTSLIDE BAR
- ENUMSET
- CHECKLIST
- TEXT or INFO

These names appear in the form BNF file syntax.

These controls use the default system colors:

'background Set background of control

'text Set text of control

The *g_color* argument is either a color symbol (for non DB options), a number for DB color options, or *nil* (for restoring to system default). See [Accessing PCB Editor Colors with AXL-SKILL](#) on page 57 for the allowed values.

Other form controls support color as a fundamental part of their interface. These are *COLOR* (See [Accessing PCB Editor Colors with AXL-SKILL](#) on page 57) and *GRID* (See [Using Grids](#) on page 470) controls.



Please note the following restrictions:

- Setting the same or close text and background colors can cause readability issues.
- Setting the background of CHECKLIST controls is not supported on UNIX.
- Dialog boxes with popups do not correctly show color.

Arguments

<i>o_form</i>	Form handle.
<i>t_field</i>	Field name.
<i>g_option</i>	Option (see above)
<i>g_color</i>	Color (see <code>axlColorDoc</code>) or nil

Value Returned

<i>t</i>	Color changed.
<i>nil</i>	Error due to an incorrect argument.

Example 1

You can find an example in `axlform.il`.

```
axlFormColorize(f1s "string" 'text 'red)
```

Sets text of string control to red.

Example 2

```
axlFormColorize(f1s "string" 'background 'green)
```

Sets background of string control to green.

Example 3

```
axlFormColorize(fls "string" 'text nil)  
axlFormColorize(fls "string" 'background nil)
```

Sets control back to default.

Example 4

```
axlFormColorize(fls "string" 'background 1)
```

Sets control background to PCB Editor database color 1

axlFormGetActiveField

```
axlFormGetActiveField(  
    r_form  
)  
⇒ t/nil
```

Description

Gets the form's active field.

Arguments

<i>r_form</i>	Form's <i>dbid</i> .
<i>t_field</i>	Form field name (string).

Value Returned

<i>t_field</i>	Active field name.
<i>nil</i>	No active field.

axlFormGridBatch

```
axlFormGridBatch(  
    r_cell  
)  
⇒ t/nil
```

Description

Always used with `axlFormGridSetBatch`. Sets many grid cells efficiently.

Arguments

r_cell	Obtained from <code>axlFormGridNewCell</code> .
--------	---

Value Returned

t	Grid cells set.
---	-----------------

nil	No grid cells set.
-----	--------------------

axlFormGridCancelPopup

```
axlFormGridCancelPopup(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

After any change to grid content, the application must tell the grid that the changes are complete. The grid then updates itself to the user. Changes include: adding or deleting columns and changing cells.

Arguments

r_form Form handle.

t_field Field name.

Value Returned

t Success.

nil Failure due to incorrect arguments.

axlFormGridDeleteRows

```
axlFormGridDeleteRows(  
    r_form  
    t_field  
    x_row  
    x_number  
)  
⇒ t/nil
```

Description

Deletes *x_number* rows at *x_row* number. *x_row=,>*, *x_number=-1* deletes the entire grid. *x_row=-1*, *x_number-1* may be used to delete the last row in the grid.

Arguments

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.
<i>x_row</i>	Row number to start delete.
<i>x_number</i>	Number of rows to delete.

Value Returned

<i>t</i>	Rows deleted.
<i>nil</i>	No rows deleted.

axlFormGridEvents

```
axlFormGridEvents(  
    r_form  
    t_field  
    s_event/(s_event1 s_event2 ...)  
)  
⇒ t/nil
```

Description

Sets user events of interest. It is critical for your application to only set the events that you actually process since enabled events are scripted.

Grid events include the following:

'rowselect	Puts grid into row select mode. This is mutually exclusive with cellselect.
'cellselect	Puts grid into cell select mode. This is mutually exclusive with rowselect.
'change	Enables cell change events. Use this option if you have check box and text box type cells.
'rightpopup	Enables right mouse button popup. A popup must have been specified in the form file.
'rightpopupPre	Enables callback to application before a right mouse popup is displayed. This allows the user to modify the popup shown. Also requires 'rightpopup be set.
'leftpopupPre	Enables callback to application before a left mouse popup is displayed. This allows the user to modify the popup shown. Left mouse popups are only present in the drop down cell type.

By default, the grid body has `rowselect` enabled while the headers have nothing enabled.

The form callback structure (*r_form*) has the following new attributes that are only applicable for grid field types:

Event	Row	Col	<Data Fields>
rowselect	<row>	1	No
cellselect	<row>	<col>	Yes (1)
change	<row>	<col>	Yes (1)
rightpopup	<row>	<col>	Yes
rightpopupPre	<row>	<col>	No (2) (3)
leftpopupPre	<row>	<col>	No (2) (3)

where:

<row> Row number (1 based)

<col> Column number (1 based)

<Data fields> Setting of the *r_form* attributes *curValue*, *curValueInt* and *isValueString*.

- Communicates the value of the data *before* the field is changed. The change event sends the value *after* the field is changed.
- Events are sent immediately before a popup is displayed so the application has the opportunity to modify it. See [axlFormGridEvents](#) on page 539 to set this and other event options.
- If using events rightpopupPre or leftpopupPre, the popup may be cancelled by calling `axlFormGridCancelPopup` when you receive one of these events.

See [Using Grids](#) on page 470 for a grid overview.

Note: Assigning events to a grid overrides the previous assignment.

This does not work:

```
axlFormGridEvents(fw "grid 'change")
axlFormGridEvents(fw "grid 'cellselect")
```

This works:

```
axlFormGridEvents(fw "grid '(cellselect change) ")
```

Arguments

r_form Form handle.

t_field Field name.

s_events See above.

Value Returned

t User event set.

nil No user event set.

axlFormGridGetCell

```
axlFormGridGetCell(  
    r_form  
    t_field  
    r_cell  
)  
⇒ r_cell/nil
```

Description

Returns grid cell data for a given row and column. All associated data for the cell is returned.

Note: The cell value is always returned as a string except for REAL and LONG data types which are returned in their native format.

If row or cell number of 0 is used then top or side heading data is returned (if present.)

Note: For best performance, reuse the cell if accessing multiple cells.

Arguments

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.
<i>r_cell</i>	Grid cell from axlFormGridNewCell().

Value Returned

<i>r_cell</i>	Cell data.
nil	Invalid form id, field label or cell doesn't exist in the grid.

Example

```
cell = axlFormGridNewCell()  
cell->row = 3  
cell->col = 1  
axlFormGridInsertRows(form, "grid" cell)  
printf("cell value = %L\n", cell)
```

Returns the value of cell - (1,3).

axlFormGridInsertCol

```
axlFormGridInsertCol(  
    r_form  
    t_field  
    r_formGridCol  
)  
⇒ t/nil
```

Description

Adds a column with the indicated options (*g_options*) to a grid field. The *g_options* parameter is based on the type `formGridCol`. The `formGridCol` structure has default behavior for all settings.

Note: For more information on using this function, see [Using Grids](#) on page 470 for an overview.

[Table 11-5](#) on page 544 describes the `FormGridCol` attributes.

Table 11-5 FormGridCol Attributes

Attribute	Type	Default	Description
<code>fieldType</code>	<code>symbol</code>	<code>TEXT</code>	Field types include: <code>TEXT</code> , <code>STRING</code> , <code>LONG</code> , <code>REAL</code> , <code>ENUMSET</code> , and <code>CHECKITEM</code> .
<code>fieldLength</code>	<code>integer</code>	16	Maximum data length.
<code>colWidth</code>	<code>integer</code>	0	Width of column.
<code>headText</code>	<code>n/a</code>	<code>n/a</code>	If the grid has a top heading, sets the heading text. Can also set using <code>axlFormGridSet</code> .

Alignment Types (`left`, `right`, and `center`):

<code>align</code>	<code>symbol</code>	<code>Left</code>	Column alignment.
<code>topAlign</code>	<code>symbol</code>	<code>Center</code>	Column header alignment.

Table 11-5 FormGridCol Attributes

Attribute	Type	Default	Description
scriptLabel	string	<row number>	Column scripting name. If the column entry can be edited, you can provide a name which is recorded to the script file. For fieldTypes of TEXT, this option is ignored. Case is ignored and text should not have whitespace or the symbol '!'.
popup	string	n/a	Name of the associated popup. May be applied to columns or cells of types ENUMSET, STRING, LONG, or REAL.
decimals	integer	n/a	Number of decimal places.
max	integer or float	n/a	Maximum value.
min	integer or float	n/a	Minimum value.

Note: Accuracy support is only applicable for LONG and REAL column types. If used, you must set both min and max values.

Note: You can add columns to a grid field only at creation time. Once rows have been added to a grid, no new columns may be added. This is true, even if you delete all rows in the grid.

Arguments

- r_form* Form handle.
- t_field* Field name.
- r_formGridCol* Instance of type `formGridCol`.

Value Returned

- t* Column added.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

nil Failure due to a nonexistent form or field, field not of type GRID, errors in the *g_options* defstruct, or grid already had a row added.

Examples

For a complete grid programming example, see: <cdsroot>/share pcb/examples/skill/ui.

Example 1

```
options = make_formGridCol
options->fieldType = 'TEXT
options->align = 'center
axlFormGridInsertCol(r_form "grid" options)
```

Adds the first column of type TEXT (non-editable) with center alignment.

Example 2

```
options->fieldType = 'ENUMSET
options->popup = "grid2nd"
options->colWidth = 10
options->scriptLabel = "class"
axlFormGridInsertCol (r_form "grid" options)
```

Adds the second column of type ENUM (non-editable) with column width of 10 and center alignment, assuming that the form file has a popup definition of *grid2nd*.

axlIsGridCellType

```
axlIsGridCellType(  
    r_cell  
)  
⇒ t/nil
```

Description

Tests the passed symbol to see if its user type is of the form "grid cell".

Arguments

<i>r_cell</i>	Symbol
---------------	--------

Value Returned

t	Symbol is of the type form grid cell.
---	---------------------------------------

nil	Symbol is not of the type form grid cell.
-----	---

axlFormGridInsertRows

```
axlFormGridInsertRows(  
    r_form  
    t_field  
    x_row  
    x_number  
)  
⇒ t/nil
```

Description

Inserts *x_number* rows at *x_row* number location. Rows are inserted empty. A -1 may be used as *x_row* to add to end of the grid. Since grids are 1 based, a 1 inserts at the top of the grid.

Arguments

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.
<i>x_row</i>	Row number of insertion point.
<i>x_number</i>	Quantity of rows to add.

Value Returned

<i>t</i>	One or more rows inserted.
<i>nil</i>	No rows inserted.

axlFormGridNewCell

```
axlFormGridNewCell(  
)  
⇒ r_cell
```

Description

Creates a new instance of *r_cell* which is required as input to `axlFormGridBatch` or `axlFormSetField` for form grid controls. As a convenience, the consuming APIs do not modify the cell attributes. You need not reset all attributes between API calls.

See [axlFormGridSetBatch](#) on page 552 for a complete description of cell attributes.

Arguments

None.

Value Returned

<i>r_cell</i>	New list gridcell handle.
---------------	---------------------------

axlFormGridReset

```
axlFormGridReset(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Resets grid to its unloaded state. Application should then set the columns, then rows, to the same state as when they initially loaded the windows.

Changes the number of columns after the grid has already been initialized.

Arguments

r_form Form handle.

t_field Field name.

Value Returned

t Grid reset.

nil Grid not reset.

Example

For a programming example, see `fgrid.il` in `<cdsroot>/share pcb/examples/skill/`

Pseudo code:

```
axlFormGridReset(fg "grid")  
initCols()  
initRows()  
axlFormGridUpdate(fg "grid")
```

axlFormGridSetBatch

```
axlFormGridSetBatch(  
    r_form  
    t_field  
    s_callback  
    g_pvtData  
)  
⇒ t/nil
```

Description

Changes grid cells much faster than `axlFormSetField` when changing multiple cells. Both APIs require a grid cell data type (`axlFormGridNewCell`.)

Grid performs single callback using `s_callback` to populate the grid. You must call `axlFormGridBatch` in the callback in order to update grid cells.

See the programming example, `grid.il` at `<cdsroot>/share pcb/examples/skill/ui`. Create rows and columns before calling this batch API.

Within the callback, use only `axlFormNewCell` and `axlFormGridBatch` from the `axlForm` API.

After changing cells, update the display using `axlFormGridUpdate` outside of the callback.

Grid Cell Data Type (r_cell) Attributes

<code>x_row</code>	Row to update.
<code>x_col</code>	Column to update.
<code>g_value</code>	Value (may be string, integer, or float) if <code>nil</code> , preserve current grid setting for the cell.
<code>sBackColor</code>	Optional background color.
<code>sTextColor</code>	Optional text color.
<code>s_check</code>	Set or clear check mark for <code>CHECKITEM</code> cells. Ignored for non-check cells. Value may be <code>t</code> or <code>nil</code> .
<code>s_noEdit</code>	If cell is editable, disables edit. Ignored for <code>TEXT</code> columns since they are not editable. Current settings are preserved.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

<code>s_invisible</code>	Make cell invisible. Current cell settings are preserved by the grid.
<code>s_popup</code>	Use popup name in the form file to set this, or "" to unset. If enum, string, long, or real cell, then overrides column popup, else restores back to popup of the column. Ignored for all other cell types.
<code>t_objType</code>	Object name "r_cell" (read-only)

Note: Previous grid cell settings are overridden by values in `s_noEdit` and `s_invisible`.

Column and Row access:

Rows and columns are 1 based. To set the cell in the first column and row, you set the row and col number to 1.

You can control header and script text with reserved row and column values as follows:

(`<row>`, 0) Set side header display text.

(`<row>`, -1) Set side header scripting text.

Note: Case is ignored, and text must not contain spaces or the '!'.

(0, `<col>`) Set top header display text. You may also set the top header at column creation time using `axlFormGridInsertCol`.

(0, 0) Setting not supported.

For headers and script text, `g_value` is the only valid attribute other than `row` and `col`.

Colors available for `s_backColor` and `s_textColor`:

- nil - use system defaults for color, typically white for background and black for text
- black
- white
- red
- green
- yellow
- button - use the current button background color

Arguments

`r_form` Form handle.

`t_field` Field name.

`t_callback` Function to callback. Takes a single argument: `g_pvtData`

`g_pvtData` Private data (Pass `nil` if not applicable.)

Value Returned

`t` Grid cell changed.

`nil` No grid cell changed, or application callback returned `nil`.

axlFormGridUpdate

```
axlFormGridUpdate(  
    r_form  
    t_field  
) -> t/nil
```

Description

Unlike the form lists control you must manually notify the grid control that it must update itself. You should use this call in the following situations:

- Inserting a row or rows
- Deleting a row or rows
- Changing cell(s)

You should make the call at the end of all of changes to the grid.



Do not make this call inside the function you use with axlFormGridSetBatch. Make it after axlFormGridSetBatch returns.

Arguments

r_form Standard form handle.

t_field Standard field name.

Value Returned

Returns *t* for success, *nil* for failure.

See also

[axlFormGridNewCell](#)

axlFormInvalidateField

```
axlFormInvalidateField(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Invalidates the form's field. Allows Windows to send a redraw message to the field's redraw procedure.

Use only for thumbnail fields.

Arguments

r_form Form handle.

t_field Field name.

Value Returned

t Field invalidated.

nil No field invalidated.

axlFormIsFieldEditable

```
axlFormIsFieldEditable(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Checks whether the given form field is editable. If the field is editable, *t* is returned. If the field is greyed, then *nil* is returned.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.

Value Returned

<i>t</i>	Field is editable.
<i>nil</i>	Field is greyed, or not editable.

axlFormListAddItem

```
axlFormListAddItem(  
    r_form  
    t_field  
    t_listItem/lt_listItems/nil  
    g_index  
)  
⇒ t/nil
```

Description

Adds an item to a list at position *x*. To add many items efficiently, pass the items as a list.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>t_listItem</i>	String of items in the list. If adding to list for the first time, you must send a <code>nil</code> to display the list.
<i>lt_listItems</i>	List of strings to add.
<i>g_index</i>	0 = First item in the list. -1 = Last item in the list.

Value Returned

<i>t</i>	One or more items added to list.
<i>nil</i>	No items added to list due to incorrect arguments.

Example 1

```
axlFormListAddItem(f1, "list" "a" -1)  
axlFormListAddItem(f1, "list" "b" -1)  
axlFormListAddItem(f1, "list" "c" -1)  
; since first time, send a nil to display the list  
axlFormListAddItem(f1, "list" nil, -1)
```

Adds three items to the end of a list.

Example 2

```
axlFormListAddItem(f1, "list" '( "a" "b" "c") , -1)
```

Adds three items to the end of a list (alternate method).

axlFormListDeleteItem

```
axlFormListDeleteItem(  
    r_form  
    t_field  
    t_listItem/x_index/lt_listItem/nil  
)  
⇒ t/x_index/nil
```

Description

Deletes indicated item in the list. You can delete by a string or by position. Deleting by string works best if all items are unique. Position can be problematic if you have the list sort the items that you add to it.

To quickly delete multiple items, call this interface with a list of items.

Note: Delete by list only supports a list of *strings*.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>x_index</i>	Position of the item to be deleted. 0 is the first item in the list, -1 is the last item in the list.
<i>t_listItem</i>	String of item to delete.
<i>lt_listItems</i>	List of items to delete.
<i>nil</i>	Deletes the last item.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

Value Returned

<code>x_index</code>	If using strings (<code>t_listItem</code>) to delete items, returns the index of strings deleted. Useful for allowing the code to automatically select the next item in the list.
<code>t</code>	If deleting by index (<code>x_index</code>), it returns <code>t</code> if successful in deleting the item.
<code>nil</code>	Failed to delete the item.

axlFormListGetItem

```
axlFormListGetItem(  
    r_form  
    t_field  
    x_index  
)  
⇒ t_listItem/nil
```

Description

Returns the item in the list at index (*x_index*.) Lists start at index 0. If -1 is passed as an index, returns the last item in the list.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>x_index</i>	Offset into the list. 0 = First item in the list. -1 =Last item in the list.

Value Returned

<i>t_listItem</i>	String of item in the list.
nil	Index not valid, or no item at that index.

axlFormListGetSelCount

```
axlFormListGetSelCount(  
    r_form  
    t_field  
)  
==> x_count/nil
```

Description

This only applies to a multi-select list box (OPTIONS multiselect in form file). Returns a count of number of items selected in a multi-select list box.

Arguments

<i>r_form</i>	Form control.
<i>t_field</i>	Name of the field.

Values Returned

<i>nil</i>	If not a multi-list box.
<i>x_count</i>	Number of items selected.

See also

`axlFormListGetSelItems`

Example

See `axlform.il` example.

axlFormListGetSelItems

```
axlFormListGetSelItems(  
    r_form  
    t_field  
)  
==> lt_selected/nil
```

Description

This only applies to a multi-select list box (OPTIONS multiselect in form file).

For a multi-select list box returns list of strings for items selected. If no items selected or this is not appropriate for control returns nil.

Arguments

r_form Form control.

t_field Name of the field.

Value Returned

lt_selected List of strings for items selected.

nil Error or nothing selected.

See also

`axlFormListGetSelCount`, `axlFormListSelAll`

Example

See `axlform.il` example.

axlFormListOptions

```
axlFormListOptions(  
    r_form  
    t_field  
    s_option/(s_option1 s_option2 ...)  
)  
⇒ t/nil
```

Description

Sets options for a list control. The following options are supported:

'doubleClick Enable double-click selection. Passing a `nil` for an option sets default list behavior. Default is single click.

Double-click events are handled as follows:

- Receive the first click as an event with the item selected and the result is:
`doubleClick = nil`.
- Receive the second click as an event with the item selected and the result is:
`doubleClick = t`.

Suggested use model:

On first click do what would normally happen if the user clicks only once. The second click is a natural extension. For example, on a browser the first click selects the file. The second click does what the *OK* button would do: send the file to the application and close the form.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>s_option</i>	Sets option for list control. <code>nil</code> resets to default.

Value Returned

- | | |
|-----|--|
| t | Options set. |
| nil | No options set due to incorrect arguments. |

Example 1

```
axlFormListOptions(form "list" 'doubleClick)
```

Enables double-click for a list.

Example 2

```
axlFormListOptions(form "list" nil)
```

Disables double-click for a list.

axlFormListSelAll

```
axlFormListSelAll(  
    r_form  
    t_field  
    g_set  
)  
==> t/nil
```

Description

This only applies to a multi-select list box (OPTIONS multiselect in form file).

This either selects or deselects all items in list box.

Arguments

<i>r_form</i>	Form control.
<i>t_field</i>	Name of the field.
<i>g_set</i>	t to select all; nil to deselect all.

Value Returned

t if successful, nil if field is not a mutli-select list box

See also

[axlFormListGetSelItems](#)

Examples

Select all items in multi-list control; mlistfield.

```
axlFormListSelAll(fw "mlistfield" t)
```

De-Select all items in multi-list control; mlistfield.

```
axlFormListSelAll(fw "mlistfield" nil)
```

axlFormMsg

```
axlFormMsg(  
    r_form  
    t_messageLabel  
    [g_arg1 ...]  
)  
⇒ t_msg/nil
```

Description

Retrieves and prints a message defined in the form file by message label (*t_messageLabel*.) Form file allows definitions of messages using the "MESSAGE" keyword (see [Using Forms Specification Language](#) on page 457.) Use this to give a user access to message text, but no access to your SKILL code.

Messages are only printed in the status area of the form owning the message (*r_form*.) You cannot access message ids from one form file and print to another. The main window is used for forms with no status lines.

You use standard formatting and argument substitution (see `printf`) for the message.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_messageLabel</i>	Message label defined in form file by the MESSAGE keyword.
<i>g_arg1</i> ...	Substitution parameters (see <code>printf</code>)

Value Returned

<i>t_msg</i>	Message that prints.
<i>nil</i>	No message with the given name found in this form file.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

Examples

```
Form file (level: 0 is info, 1 is info with no journal entry, 2 is warning, 3  
is error, and 4 is fatal.);  
    MESSAGE drccount 0 "Drc Count of %d for %s"  
    MESSAGE drcerrors 2 "Drc Errors"  
  
axlFormMsg(fw "drccount" 10 "spacing")  
axlFormMsg(fw "drcerrors")
```

axlFormGetFieldType

```
axlFormGetFieldType(  
    r_form  
    t_field  
)  
⇒ g_fieldType/nil
```

Description

Returns the control type for a form field. See the keywords in [Callback Procedure: formCallback](#) on page 526 for a list of supported field types.

Arguments

r_form Form *dbid*.

t_field Field name.

Value Returned

g_fieldType One of the control types.

nil Field does not exist or is not one of the types supported.

axlFormDefaultButton

```
axlFormDefaultButton(  
    r_form  
    t_field/g_mode  
)  
⇒ t/nil
```

Description

Forms normally automatically set a *default button* in a form with the DEFAULT section in the form file or with the *OK* and *DONE* labels. When the user hits a carriage return, the *default button* is executed.

A form can have, at most, one default button. Only a field of type **BUTTON** can have the default button attribute.

Note: If default buttons are disabled in a form, then attempts to establish a new default button are ignored. You can only change the default button if the capability in the form is enabled.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Field name to establish as new button default.
<i>g_mode</i>	<i>t</i> to enable the default button in the form, <i>nil</i> to disable it.

Value Returned

<i>t</i>	Default button set.
<i>nil</i>	Field does not exist.

Example 1

```
axlFormDefaultButton(form nil)
```

Sets no default button in form.

Example 2

```
axlFormDefaultButton(form "cancel")
```

Sets the default button to be Cancel instead of the default OK.

axlFormGridOptions

```
axlFormGridOptions(  
    r_form  
    t_field  
    s_name  
    [g_value]  
)  
⇒ t/nil
```

Description

Miscellaneous grid options. See [Using Grids](#) on page 470 for a grid overview.

Arguments

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.
<i>s_name/g_value</i>	Supported options shown.

s_name/g_value Supported Options

['goto x_row']	Puts the indicated row on display, scrolling the grid if necessary.
----------------	---

Note: -1 signifies the last row.

['goto x_row:x_col']	Sends grid to indicated row and column.
----------------------	---

Note: -1 signifies the last row or column.

['select x_row']	Selects (highlights) indicated row.
------------------	-------------------------------------

['select x_row:x_col']	Selects (highlights) indicated cell. Grid must be in cell select mode else row is selected instead of cell. See <code>axlFormGridEvents</code> for more information.
------------------------	--

['deselectAll']	Deselect any selected grid cells or rows.
-----------------	---

Value Returned

t	Selected grid option performed.
nil	Selected grid option not performed.

Example 1

```
axlFormGridOption(fw, "mygrid" 'goto 10)
```

Makes row 10 visible.

Example 2

```
axlFormGridOption(fw, "mygrid" 'goto 5:2)
```

Makes row 5 column 2 visible.

Example 3

```
axlFormGridOption(fw, "mygrid" 'deselectAll)
```

Deselects anything highlighted in the grid.

axlFormSetActiveField

```
axlFormSetActiveField(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Makes the indicated field the active form field.

Note: If you do an `axlFormRestoreField` in your dispatch handler on the field passed to your handler, then that field remains active.

Arguments

r_form Form *abid*.

t_field Field name.

Value Returned

t Field set active.

nil Failed to set the field active.

axlFormSetDecimal

```
axlFormSetDecimal(  
    o_form  
    g_field  
    x_decimalPlaces  
)  
⇒ t/nil
```

Description

Sets the decimal precision for real fill-in fields in the form. If *g_field* is *nil*, sets the precision for all real fill-in fields in the form.

Arguments

<i>o_form</i>	Form handle.
<i>g_field</i>	Field label, or <i>nil</i> for all fields.
<i>x_decimalPlaces</i>	Number of decimal places - must be a positive integer.

Value Returned

<i>t</i>	Successfully set new decimal precision.
<i>nil</i>	Error due to invalid arguments.

axlFormSetFieldEditable

```
axlFormSetFieldEditable(  
    r_form  
    t_field  
    g_editable  
)  
⇒ t/nil
```

Description

Sets individual form fields to editable (t) or greyed (nil).

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Field name.
<i>g_editable</i>	Editable (t) or greyed (nil).

Value Returned

t	Set form field to editable or greyed.
nil	Failed to set form field to editable or greyed.

axlFormSetFieldLimits

```
axlFormSetFieldLimits(  
    o_form  
    t_field  
    g_min  
    g_max  
)  
⇒ t/nil
```

Description

Sets the minimum or maximum values a user can enter in an integer or real fill-in field. If a nil value is provided, that limit is left unchanged.

For a REAL field, the type for *g_min* and *g_max* may be int, float, or nil. For an INT or LONG, the type must be int or nil.

Arguments

<i>o_form</i>	Form handle.
<i>t_field</i>	Field label.
<i>g_min</i>	Minimum value for field.
<i>g_max</i>	Maximum value for field.

Value Returned

<i>t</i>	Set max or min.
nil	Error due to invalid argument(s).

axlFormTreeViewAddItem

```
axlFormTreeViewAddItem(  
    r_form  
    t_field  
    t_label  
    g_hParent  
    g_hInsertAfter  
    [g_multiSelectF]  
    [g_hLeafImage]  
    [g_hOpenImage]  
    [g_hClosedImage]  
)  
⇒ g_hItem/nil
```

Description

Adds an item to a treeview under *parent* and after *insertAfter* sibling. If sibling is *nil*, item is added as the last child of a parent. If parent is *nil*, item is created as the root of the tree.

Note: You must use this to add an item to a tree. *axlFormSetField* is disabled for Tree controls.

Applications must keep the returned handle *l_hItem* since a handle will be passed as *form->curValueInt* when the item is selected from tree view. The string associated with the selected item is also passed as *form->curValue*, however the string value may not be unique and cannot be used as a reliable identifier for the selected treeview item.

Note: Tree view defaults to single selection mode. There is no checkbox associated with items in the tree view to make multiple selections. To make a tree view item multi select, you pass one of the following values for *t_multiSelectF*:

- *nil* or 'TVSELECT_SINGLE for no selection state checkbox
- *t* or 'TVSELECT_2STATE for 2 state checkbox
- 'TVSELECT_3STATE for 3 state checkbox

If an item is defined as multi select, a check box appears as part of the item. The user can check/uncheck (2 state) this box to indicate selection or select checked/unchecked/disabled modes for a 3 state checkbox. When the user makes any selection in the checkbox, its value is passed to application code in *form->treeViewSelState*. In this case, *form->curValue* is *nil*.

Arguments

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Field name.
<i>t_label</i>	String of item in the treeview.
<i>g_hParent</i>	Handle of parent. If <code>null</code> , item created as the root of the tree.
<i>g_hInsertAfter</i>	Handle of the sibling to add the item after. If <code>null</code> , item added at the end of siblings of the parent.
<i>t_multiSelectF</i>	If <code>t</code> , the item has a checkbox for multi selection.
<i>g_hLeafImage</i>	Handle of the image to use whenever a leaf node in the tree view. If <code>nil</code> or not supplied, the default pink diamond image is used.
<i>g_hOpenImage</i>	Handle of image to use whenever an expanded parent node in the tree view. If <code>nil</code> or not supplied, the default open folder image is used.
<i>g_hClosedImage</i>	Handle of image to use whenever an unexpanded parent node in the tree view. If <code>nil</code> or not supplied, the default closed folder image is used.

Value Returned

<i>g_hItem</i>	Item is added to the tree view control.
<code>nil</code>	No item is added to the tree view control due to an error.

Examples

Creates a treeview.

Constructs a list that resembles a tree structure. Each list element is composed of the item label and the handle that was returned from a call to `axlFormTreeViewAddItem()`. The structure is for the following tree sample:

```
"For Sale"
  "Redmond"
    "100 Berry Lane"
    "523 Apple Road"
    "1212 Peach Street"
  "Bellevue"
    "22 Daffodil Lane"
    "33542 Orchid Road"
    "64134 Lily"
  "Seattle"
    "33 Nicholas Lane"
    "555 Tracy Road"
    "446 Jean Street"

(defstruct myTreeItem string id)

(defun myTreeItems ()
(list
(make_myTreeItem ?string "For Sale" ?id nil)
(list (make_myTreeItem ?string "Redmond" ?id nil)
      (list (make_myTreeItem ?string "100 Berry Lane" ?id nil))
      (list (make_myTreeItem ?string "523 Apple Road" ?id nil))
      (list (make_myTreeItem ?string "12 Peach Street" ?id nil)))
(list (make_myTreeItem ?string "Bellevue" ?id nil)
      (list (make_myTreeItem ?string "22 Daffodil Lane" ?id nil))
      (list (make_myTreeItem ?string "354 Orchid Road" ?id nil))
      (list (make_myTreeItem ?string "64 Lily Street" ?id nil)))
(list (make_myTreeItem ?string "Seattle" ?id nil)
      (list (make_myTreeItem ?string "33 Nicholas Lane" ?id nil))
      (list (make_myTreeItem ?string "5 Tracy Road" ?id nil))
      (list (make_myTreeItem ?string "46 Jean Street" ?id nil)))))

(defun axlAddTreeItem (form field parent item)
```

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

```
(let (hParent hItem hAfter str)
  (cond
    ((get item 'string)
     (setq str (get item 'string))
     (setq hParent (get parent 'id))
     (setq hAfter nil) ; for now
     (setq hItem (axlFormTreeViewAddItem form field str hParent hAfter))
     item->id = hItem
     (t
      nil)))))

(defun axlAddTreeItems (form field l_list)
  (let ((parent (car l_list)))

    ;; root node
    (axlAddTreeItem form field nil parent)

    ;; Add the rest of the nodes
    (axlAddTreeItemsRecursive form field l_list)))

(defun axlAddTreeItemsRecursive (form field l_list)
  (let ((parent (car l_list)))
    (when parent
      (mapc
        (lambda (item)
          (axlAddTreeItem form field parent (car item))
          (axlAddTreeItemsRecursive form field item))
        (cdr l_list))))
```

Callback Values

In your callback function, the form has the following properties relevant to treeviews:

form->curValue Contains label of a treeview item. Set in single select mode and in multi select mode when the user selects the item. In this case, the *result->tree.selectState* is -1.

form->curIntValue Contains id of the selected treeview item.

form->selectState In multi select mode, when the user picks the selection checkbox, this field contains:

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

- 0 if selection checkbox is not checked
- 1 if selection checkbox is checked
- 2 if selection checkbox is disabled

In this case the *result->string* is empty. In all other cases the value is -1.

form->event

If event property is set to "rightpopup", treeview control has a popup and user has selected an item in the popup. In this case, *form->curValue* is set to the popup index selected, and *form->selectState* is set to -1. *form->curIntValue* is set to the treeview item id.

You add popups to treeview fields in a form like you add any other field in a form.

For all non-popup operations, event is set to "normal."

axlFormTreeViewChangeImages

```
axlFormTreeViewChangeImages(  
    r_form  
    t_field  
    g_hItem  
    [g_hLeafImage]  
    [g_hOpenImage]  
    [g_hClosedImage]  
)  
⇒ t/nil
```

Description

Modifies various bitmap images associated with a given tree view item.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>g_hItem</i>	Handle of item in tree view. Handle was returned as a result of the call to <code>axlFormTreeViewAddItem</code> when item was initially added.
<i>g_hLeafImage</i>	Handle of the image to use whenever item is a leaf node in the tree view. If <code>nil</code> or not supplied, the default pink diamond image is used.
<i>g_hOpenImage</i>	Handle of image to use whenever item is an expanded parent node in the tree view. If <code>nil</code> or not supplied, the default open folder image is used.
<i>g_hClosedImage</i>	Handle of image to use whenever item is an unexpanded parent node in the tree view. If <code>nil</code> or not supplied, the default closed folder image is used.

Allegro PCB and Package User Guide: SKILL Reference

Form Interface Functions

Value Returned

t Tree view item's images are modified.

nil Failed to modify tree view item's images.

axlFormTreeViewChangeLabel

```
axlFormTreeViewChangeLabel(  
    r_form  
    t_field  
    g_hItem  
    t_label  
)  
⇒ t/nil
```

Description

Modifies text of a given treeview item.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>g_hItem</i>	Handle of item in the tree view. This handle was returned as a result of the call to <code>axlFormTreeViewAddItem</code> .
<i>t_label</i>	New label.

Value Returned

<i>t</i>	Tree view item's label is modified.
<i>nil</i>	Failed to modify tree view item's label.

axlFormTreeViewGetImages

```
axlFormTreeViewGetImages(  
    r_form  
    t_field  
    g_hItem  
)  
⇒ l_hImage/nil
```

Description

Returns various bitmap image handles that refer to images used by a specified item in the tree view.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>g_hItem</i>	Handle of item in the tree view. This handle was returned as a result of the call to <code>axlFormTreeViewAddItem</code> when this item was initially added.

Value Returned

<i>l_hImage</i>	List of three image handles. The first is the handle of the image used when this item is a leaf node. The second is the handle of the image used when this item is an expanded parent node. The third is the handle of the image used when this item is an unexpanded parent node.
<i>nil</i>	Error due to invalid arguments.

axlFormTreeViewGetLabel

```
axlFormTreeViewGetLabel(  
    r_form  
    t_field  
    g_hItem  
)  
⇒ t_label/nil
```

Description

Returns text of a given treeview item.

Arguments

r_form Form id.

t_field Field name.

g_hItem Handle of item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem` when this item was initially added.

Value Returned

t_label Text of given tree view item.

nil Failed to get text of given tree view item due to invalid arguments.

axlFormTreeViewGetParents

```
axlFormTreeViewGetParents(  
    r_form  
    t_field  
    g_hItem  
)  
⇒ lg_hItem/nil
```

Description

Returns a list of all the ancestors of a treeview control item, starting from the root of the tree. Helps in search operations in SKILL. Applications can traverse their tree list following parent lists to a given item instead of searching the whole tree for an item.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>g_hItem</i>	Handle of item in the tree view. This handle was returned as a result of the call to <code>axlFormTreeViewAddItem</code> when this item was initially added.

Value Returned

<i>lg_hItem</i>	List containing all parents, starting from the root.
<i>nil</i>	Failed to obtain list due to invalid arguments.

axlFormTreeViewGetSelectState

```
axlFormTreeViewGetSelectState(  
    r_form  
    t_field  
    g_hItem  
)  
⇒ x_selectState
```

Description

In multi select mode, returns the select state. This is different than the current selected item in single select tree views. In multi select mode, users can change the select state by clicking on the select checkbox associated with each item.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>g_hItem</i>	Handle of item in the tree view. This handle was returned as a result of the call to <code>axlFormTreeViewAddItem</code> when this item was initially added.

Value Returned

In multi select mode:

0	Checkbox is unchecked.
1	Checkbox is checked.
2	Checkbox is disabled.
-1	Single select mode or failure due to invalid arguments.

axlFormTreeViewLoadBitmaps

```
axlFormTreeViewLoadBitmaps(  
    r_form  
    t_field  
    lt_bitmaps  
)  
⇒ l_hImage/nil
```

Description

Allows an application to load one or more bitmaps into PCB Editor for use in specified tree view.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	Field name.
<i>lt_bitmaps</i>	Either a string containing the name of the bitmap file to load, or a list of strings, each of which is the name of a bitmap file to load.

Notes:

- Bitmap images must be 16 x 16 pixels.
- A bitmap file may contain more than one image provided they are appended horizontally (for example, a bitmap file containing *n* images will be $(16*n)$ x 16 pixels).
- Color RGB (255,0,0) is reserved for the transparent color. Any pixel with this color is displayed using the background color.

Value Returned

<i>l_hImage</i>	List of image handles that the caller will use to reference the images in subsequent <code>axlFormTreeViewAddItem</code> calls. List is ordered to correspond with the order that the images were listed in the <i>lt_bitmaps</i> parameter.
<i>nil</i>	One or more of the bitmap files could not be found, or an error was encountered while adding images.

Example

Given the file `myBmp1.bmp` is a 16 x 16 bitmap and `myBmp2.bmp` is a 32 x 16 bitmap.

```
(axlFormTreeViewLoadBitmaps (list "myBmp1" "myBmp2"))
```

This might return the following:

```
(6 7 8)
```

You can interpret the returned handles as follows:

- Image handle 6 refers to `myBmp1.bmp`.
- Image handle 7 refers to the left half of `myBmp2.bmp`.
- Image handle 8 refers to the right half of `myBmp2.bmp`.

axlFormTreeViewSet

```
axlFormTreeViewSet(  
    r_form  
    t_field  
    s_option  
    g_hItem  
    [g_data]  
)  
⇒ t/nil
```

Description

Allows an application to change global and individual items in a tree view control.

Arguments

r_form Form id.

t_field Field name.

s_option and *g_hItem* *s_option* is one of the symbols listed. Each is paired with *g_hItem*, the handle of an item in the tree view control. If the option you are setting is global, you use *nil* in place of *g_hItem*.

- ❑ TV_REMOVEALL (*nil*)
- ❑ TV_DELETEITEM (*g_hItem*)
- ❑ TV_ENSUREVISIBLE (*g_hItem*)
- ❑ TV_EXPAND (*g_hItem*)
- ❑ TV_COLLAPSE (*g_hItem*)
- ❑ TV_SELECTITEM (*g_hItem*, or pass *nil* to deselect item)
- ❑ TV_SORTCHILDREN (*g_hItem*)
- ❑ TV_NOEDITLABEL (*nil* - disables in place label editing)
- ❑ TV_NOSELSTATEDISPATCH (*nil* -check box selection not dispatched)
- ❑ TV_ENABLEEDITLABEL (*nil* - enable in place label editing)

- ❑ TV_MULTISELTYPE (*g_hItem*)
- ❑ TV_NOSHOWSELALWAYS (nil)
- ❑ TV_SHOWSELALWAYS (nil)

Note: For TV_MULTISELTYPE, you can also use the option *g_data* which is one of the following: TVSELECT_SINGLE, TVSELECT_2STATE, TVSELECT_3STATE. Default is TVSELECT_SINGLE.

g_hItem Handle of item in the tree view control.

Note: You get *g_hItem* in the following ways:

- Call axlFormTreeViewAddItem.
- Call axlFormTreeViewGetParents.
- Change a tree control that causes a form dispatch. Then the form User Type attributes curValue and curValueInt are the *g_hItem*.
- You can pass nil for *g_hItem* in some cases. Pass nil for TV_SELECTITEM option to deselect the item that is currently selected.

Value Returned

t Changed one or more items in tree view control.

nil Failed to change items in tree view control.

Example

```
(axlFormTreeViewSet form form->curField 'TV_DELETEITEM form->curValue)
```

Deletes selected treeview item in a form.

For more examples see <cdsroot>/share pcb/examples/skill/form/basic

axlFormTreeViewSetSelectState

```
axlFormTreeViewSetSelectState(  
    r_form  
    t_field  
    g_hItem  
    g_state  
)  
⇒ t/nil
```

Description

In multi select mode, sets the select state. This is different than the current selected item in single select tree views. In multi select mode, users can change the select state by clicking on the select checkbox associated with each item.

Arguments

<i>r_form</i>	Form id.
<i>t_field</i>	The name of the field.
<i>g_hItem</i>	Handle of the item in the tree view. This handle was returned as a result of the call to <code>axlFormTreeViewAddItem</code> when this item was initially added.
<i>g_state</i>	Select state to set. <ul style="list-style-type: none">❑ Select state is unchecked if <i>g_state</i> is nil or 'TVSTATE_UNCHECKED❑ Select state is checked if <i>g_state</i> is t or 'TVSTATE_CHECKED❑ Select state is disabled if <i>g_state</i> is 'TVSTATE_DISABLED

Value Returned

<i>t</i>	Set select state.
nil	Failed to set select state.

Allegro PCB and Package User Guide: SKILL Reference
Form Interface Functions

Simple Graphics Drawing Functions

Overview

This chapter describes the AXL-SKILL functions related to Simple Graphics Drawing. You use these drawing utilities for drawing into bitmap areas such as thumbnails within the UIF forms package.

`axlGRP` is the AXL interface to a Simple Graphics Drawing utility.

You can simplify application drawing into thumbnails within forms as follows:

1. Specify the thumbnail field within the form file. This should not have a bitmap associated with it.
2. Call the `axlGRPDrwInit` function with the form, field name, and a callback function. Keep the handle returned by this function so you can use it in later processing.
3. Using the functions provided, redraw the image. The callback function is invoked with the graphics handle as the parameter.
4. Use the `axlGRPDrwUpdate` function to trigger the callback function.

Note: The application *cannot* set bitmaps or graphics callbacks using the facilities outlined in the Thumbnail documentation while using this package.

Simple Graphics Drawing package supports the following:

- Rectangles (Filled and unfilled)
- Polygons (Filled and unfilled)
- Circles (Filled and unfilled)
- Simple Lines
- Poly Lines
- Bitmaps
- Text

This package supports a mappable coordinate system. With the `GRPDrwMapWindow` function, you can specify a rectangle that gets mapped to the actual drawing area. An aspect ratio of 1 to 1 is maintained.

The zero point of the drawing area is the upper left point of the drawing:



Note: Objects drawn earlier are overlapped by objects drawn later. The application must manage this.

Setting Option Properties on the *r_graphics* Handle

You can set option properties on the *r_graphics* handle before calling the drawing functions. These properties, if applicable, are used by the drawing properties.

Table 12-1 r_graphics Option Properties

Option	Default	Description	Available Settings
color	black	Applies to all elements.	black, white, red, green, yellow, blue, lightblue, rose, purple, teal, darkpink, darkmagenta, aqua, gray, olive, orange, pink, beige, navy, violet, silver, rust, lime, brown, mauve, magenta, lightpink, cyan, salmon, peach, darkgray, button (represents the current button color)
fill	unfilled	Applies only to rectangles and polygons.	filled, filled_solid, unfilled
width	0	Applies only when geometric elements are unfilled.	
text_align	left	Text justification. Applies only to text.	left, center, right
text_bkmode	transparent	Text background display mode. Applies only to text.	transparent, opaque

Examples

Allegro PCB and Package User Guide: SKILL Reference

Simple Graphics Drawing Functions

See `<cdsroot>/share pcb/examples/skill/form/basic/axlform.il` for a complete example.

Functions

axlGRPDrwBitmap

```
axlGRPDrwBitmap(  
    r_graphics  
    t_bitmap  
)  
⇒ t/nil
```

Description

Loads a bitmap into the drawing area in the graphics field. More drawing can take place on top of the bitmap.

Arguments

r_graphics Graphics handle.

t_bitmap Name of bitmap file. File must be on the BMPPATH, with .bmp as the extension.

Value Returned

t Bitmap loaded into drawing area in the graphics field.

nil No bitmap loaded into the drawing area in the graphics field due to invalid arguments.

axlGRPDrwCircle

```
axlGRPDrwCircle(  
    r_graphics  
    l_origin  
    x_radius  
)  
⇒ t/nil
```

Description

Draws a circle into the area identified by the *r_graphics* handle, at the origin specified, and with the specified radius. Option properties attached to the *r_graphics* handle are applied when drawing the circle.

Arguments

<i>r_graphics</i>	Graphics handle.
<i>l_origin</i>	List noting the <i>x</i> and <i>y</i> coordinates of the origin.
<i>x_radius</i>	Integer noting the radius of the circle.

Value Returned

t	Circle drawn.
nil	No circle drawn due to invalid arguments.

axlGRPDrwInit

```
axlGRPDrwInit(  
    r_form  
    t_field  
    t_func  
)  
⇒ r_graphics/nil
```

Description

Sets up necessary data structures for triggering the graphics callback into the graphics field.

Arguments

<i>r_form</i>	Handle of the form.
<i>t_field</i>	Name of field into which the package should draw. (Only THUMBNAIL fields are supported.)
<i>t_func</i>	Name of the drawing callback function. Callback function is invoked with the graphics handle as the parameter.

Value Returned

<i>r_graphics</i>	Graphics package handle.
nil	Failed to set up necessary data structures for triggering the graphics callback due to invalid arguments.

axlGRPDrwLine

```
axlGRPDrwLine(  
    r_graphics  
    l_vertices  
)  
⇒ t/nil
```

Description

Draws a line into the area identified by the *r_graphics* handle and the list of coordinates. Option properties attached to the *r_graphics* handle are applied when drawing the line.

Arguments

<i>r_graphics</i>	Graphics handle.
<i>l_vertices</i>	List of coordinates describing the line.

Value Returned

t	Line drawn.
nil	No line drawn due to invalid arguments.

axlGRPDrwMapWindow

```
axlGRPDrwMapWindow(  
    r_graphics  
    x_hgt  
    x_width  
)  
⇒ t/nil
```

Description

Allows the application to denote the coordinate system that is mapped into the drawing area of the graphics field.

Arguments

<i>r_graphics</i>	Graphics handle.
<i>x_hgt</i>	Height of drawing window.
<i>x_width</i>	Width of drawing window.

Value Returned

t	Successful
nil	Error occurred due to invalid arguments.

axlGRPDrwPoly

```
axlGRPDrwPoly(  
    r_graphics  
    l_vertices  
)  
⇒ t/nil
```

Description

Draws a polygon into the area identified by the *r_graphics* handle and the list of coordinates. Option properties attached to the *r_graphics* handle are applied when drawing the polygon. If the coordinates do not form a closed polygon, the first and last coordinates in the list are connected by a straight line.

Arguments

<i>r_graphics</i>	Graphics handle.
<i>l_vertices</i>	List of coordinates describing the line.

Value Returned

t	Polygon or line drawn.
nil	No polygon or line drawn due to invalid arguments.

axlGRPDrwRectangle

```
axlGRPDrwRectangle(  
    r_graphics  
    l_upper_left  
    l_lower_right  
)  
⇒ t/nil
```

Description

Draws a rectangle into the area identified by the *r_graphics* handle and the *upper_left* and *lower_right* coordinates. Option properties attached to the *r_graphics* handle are applied when drawing the rectangle.

Arguments

<i>r_graphics</i>	Graphics handle.
<i>l_upper_left</i>	List noting the coordinate of the upper left point of the rectangle.
<i>l_lower_right</i>	List noting the coordinate of the lower right point of the rectangle.

Value Returned

t	Rectangle drawn.
nil	No rectangle drawn due to incorrect arguments.

axlGRPDrwText

```
axlGRPDrwText(  
    r_graphics  
    l_origin  
    t_text  
)  
⇒ t/nil
```

Description

Draws text into the area identified by the *r_graphics* handle at the origin specified. Option properties attached to the *r_graphics* handle are applied when drawing the text.

Arguments

<i>r_graphics</i>	Graphics handle.
<i>l_origin</i>	List noting the <i>x</i> and <i>y</i> coordinate of the origin.
<i>t_text</i>	Text string to be drawn.

Value Returned

<i>t</i>	Text drawn.
<i>nil</i>	No text drawn due to incorrect arguments.

axlGRPDrwUpdate

```
axlGRPDrwUpdate(  
    r_graphics  
)  
⇒ t/nil
```

Description

Triggers calling of the application supplied callback function.

Arguments

r_graphics Graphics handle.

Value Returned

t Application supplied callback function called.

nil No callback function called due to an incorrect argument.

Allegro PCB and Package User Guide: SKILL Reference
Simple Graphics Drawing Functions

Message Handler Functions

Overview

This chapter describes the AXL-SKILL message handler system and functions. The message handler system allows you to write AXL-SKILL code that does not have to deal explicitly with errors after each call to a lower level routine, but rather checks at only one or two points. This contrasts with application programs that do not have a buffering and exception-handling message facility, where you must test for and respond to errors and exceptions at each point of possible occurrence in your code. Using the functions described in this chapter, you can do the following:

- Establish a *context* for your application messages.

A context is a logical section of your application during which you want to buffer and test for the potential errors and warnings you provided for in the lower levels of your application code.

- Write messages to the user with one of five levels of *severity*:

Severity Level	Type	Type Description and Response by AXL Message Handler
0	Text	Data created by the application, such as a log or report. Writes to journal and the user interface without being buffered.
1	Info	Such as “10% done,” “20% done”... Writes to user interface only, <i>not</i> to context message buffer nor to the journal.
2	Warning	Such as “Connect line is 5 mils wider than allowed.” Writes to context message buffer and journal with a “W” warning tag.

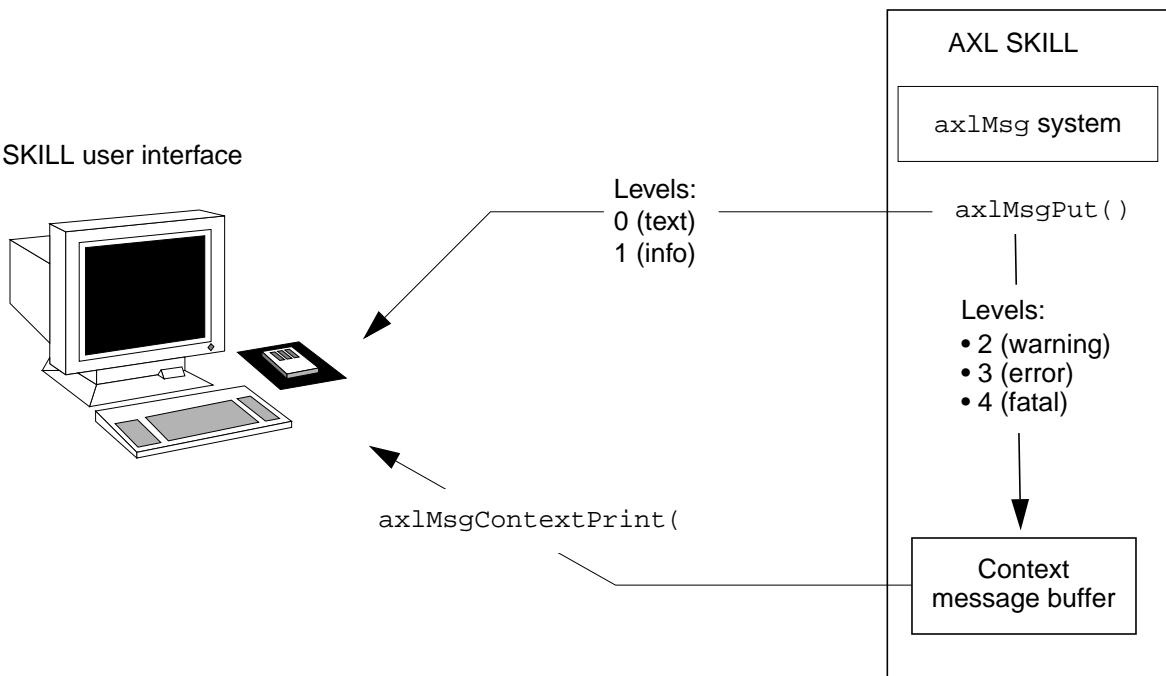
Allegro PCB and Package User Guide: SKILL Reference

Message Handler Functions

Severity Level	Type	Type Description and Response by AXL Message Handler
3	Error	Such as "Cannot find symbol DIP24_200 in library." Beeps and writes to context message buffer and journal with an "E" error tag.
4	Fatal	Such as "Disk read error. Cannot continue." Double beeps and writes to context message buffer and journal with an "F" fatal tag.

- Test and change the severity level of the messages created and buffered in a context.
- Check for specific messages in the message buffer, and respond appropriately to anticipated conditions detected by lower-level functions.
- Close a context.

The following illustration shows how `axlMsg` functions move messages to and from a context message buffer and the SKILL user interface.



Message Handler Example

```
context = axlMsgContextStart("My own context.")
axlMsgPut(list("My warning" 2))
axlMsgPut(list("My error" 3))
printf("Message severity %d",axlMsgContextTest(context))
axlMsgPut(list("My fatal error %s" 4) "BAD ERROR")
if( axlMsgContextInBuf(context "My error")
    printf("%s\n" "my error is there"))
printf("Message severity %d",axlMsgContextTest(context))
axlMsgContextPrint(context)
axlMsgContextFinish(context)
    => t
```

1. Starts a context with `axlMsgContextStart`
2. Puts a warning, an error, and a fatal error message using `axlMsgPut`
3. Checks for the error message with `axlMsgContextInBuf`
4. Tests for the context severity level with `axlMsgContextTest`
5. Prints the context buffer with `axlMsgContextPrint`
6. Ends with `axlMsgContextFinish`

When you load the SKILL program shown above, the SKILL command line outputs the following:

```
W- My warning
E- My error
F- My fatal error BAD ERROR
Message severity 3
my error is there
Message severity 4
t
```

General usage of the `axlMsg` System

- Messages first go to the context buffer
- `axlMsgContextPrint` prints them to the SKILL command line
- The contents of the output buffer from any `print` and `printf` data write to the command line when control returns to the command line. That is why the messages "Message severity 3," "my error is there" and "Message severity 4" follow the buffered messages ("W- My warning" ...)

Message Handler Functions

This section lists message handler functions.

axlMsgPut

```
axlMsgPut(  
    g_message_format  
    [g_arg1 ...]  
)  
⇒ t
```

Description

Puts a message in the journal file. Use this function to “print” messages. It buffers any *errors* or *warnings*, but processes other message classes immediately.

Arguments

<i>g_message_format</i>	Context message (<code>printf</code> like) format string. See “Overview” on page 611 for a description of messages and valid arguments.
<i>g_arg1</i> ...	Values for substitution arguments for the format string.

Value Returned

t	Always returns t.
---	-------------------

Example

See the [“Message Handler Example”](#) on page 613.

```
axlMsgPut(list( "Cannot find via %s" 3) "VIA10")  
⇒ t
```

axlMsgContextPrint

```
axlMsgContextPrint(  
    r_context  
)  
⇒ t
```

Description

Prints the buffered messages and removes them from the message buffer.

Arguments

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . Prints messages only for this context (or any children). An argument of <code>nil</code> causes <code>axlMsgContextPrint</code> to look through all contexts.
------------------	---

Value Returned

t	Always returns <code>t</code> .
---	---------------------------------

Example

See the “[Message Handler Example](#)” on page 613 in the beginning of this chapter.

```
axlMsgContextPrint(context)  
⇒ t
```

Prints the buffered messages in that context to SKILL command line:

```
W- My warning  
E- My error  
F- My fatal error BAD ERROR
```

axlMsgContextGetString

```
axlMsgContextGetString(  
    r_context  
)  
⇒ lt_messages/nil
```

Description

Gets the messages in the message buffer and removes them from the buffer. Call `axlMsgContextGetString` subsequently to communicate those messages to the user (for example, in a log file).

Arguments

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . Gets messages only for this context (or any children). An argument of <code>nil</code> causes <code>axlMsgContextGetString</code> to look through all contexts.
------------------	---

Value Returned

<i>lt_messages</i>	List of the text strings of the buffered messages.
<code>nil</code>	No buffered messages found.

Example

See the “[Message Handler Example](#)” on page 613.

```
axlMsgContextGetString(context)  
⇒ ("My warning" "My error" "My fatal error BAD ERROR")
```

axlMsgContextGet

```
axlMsgContextGet(  
    r_context  
)  
⇒ lt_format_strings/nil
```

Description

Gets the format strings of the buffered messages. (*Not* the messages themselves. Compare the example here and the one shown for `axlMsgContextGetString`.) Does not remove the messages from the message buffer, rather it simply provides the caller an alternative to making a number of `axlMsgContextInBuf` calls.

Arguments

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . Gets messages only for this context (or any children). An argument of <code>nil</code> causes <code>axlMsgContextGet</code> to look through all contexts.
------------------	---

Value Returned

lt_format_strings List of format strings of the buffered messages.

`nil` No buffered messages found.

Example

See the “[Message Handler Example](#)” on page 613.

```
mylist = axlMsgContextGet(context)  
⇒ ("My warning" "My error" "My fatal error %s")
```

axIMsgContextTest

```
axIMsgContextTest(  
    r_context  
)  
⇒ x_class
```

Description

Returns the most severe message class of the messages in the context message buffer. See [axIMsgContextInBuf](#) on page 619 to check for a particular message class.

Arguments

<i>r_context</i>	Context handle from <code>axIMsgContextStart</code> . Looks only for messages for this context. If <i>r_context</i> is nil, <code>axIMsgContextTest</code> looks through all contexts.
------------------	--

Value Returned

<i>x_class</i>	The most severe message class of the messages in the message buffer of the given context.
----------------	---

Example

See the [Message Handler Example](#) on page 613.

```
printf("Message severity %d\n" axIMsgContextTest(context))  
⇒ Message severity 4
```

axlMsgContextInBuf

```
axlMsgContextInBuf(  
    r_context  
    t_format_string  
)  
⇒ t
```

Description

Checks whether message *t_format_string* is in the message buffer of context *r_context*. Gives the application the ability to control code flow based on a particular message reported by a called function. The check is based on the original format string, not the fully substituted message.

Arguments

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . Looks only for messages in this context (or any children). If <i>r_context</i> is <code>nil</code> , <code>axlMsgContextInBuf</code> looks through all contexts.
<i>t_format_string</i>	Format string of the message.

Value Returned

<code>t</code>	Specified message is in the buffer.
<code>nil</code>	Specified message is not in the buffer.

Example

See the [“Message Handler Example”](#) on page 613.

```
if( axlMsgContextInBuf(context "My error"  
    printf(%s\n" "My error is there"))
```

axlMsgContextRemove

```
axlMsgContextRemove(  
    r_context  
    t_format_string  
)  
⇒ t
```

Description

Removes a message (or messages) from the buffered messages. Lets you remove messages (usually warnings) from the buffer that you decide, later in a procedure, that you do not want the user to see.

Arguments

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . Removes messages for only this context (or any children). If <i>r_context</i> is <code>nil</code> , <code>axlMsgContextRemove</code> looks through all contexts.
<i>t_format_string</i>	Format string of the message. The match for the message in the buffer ignores the substitution parameters used to generate the full text of the message.

Value Returned

<i>t</i>	Always returns <i>t</i> .
----------	---------------------------

Example

See the [“Message Handler Example”](#) on page 613.

```
axlMsgContextRemove(context, "My fatal error %s")  
⇒ t
```

axlMsgContextStart

```
axlMsgContextStart(  
    g_format_string  
    [g_arg1 ...]  
)  
⇒ r_context
```

Description

Indicates the start of a message context. Prints any buffered messages for the context.

Arguments

g_format_string Context message (`printf` like) message format string.

g_arg1 ... Values for the substitution arguments for the format string. Use `axlMsgClear` (and Test/Set) functions to control code flow if the function returns insufficient values.

Value Returned

r_context Context handle to use in subsequent `axlMsgContext` calls.

Example

See the [“Message Handler Example”](#) on page 613.

```
context = axlMsgContextStart("Messages for %s" "add line")  
Messages for add line  
5
```

axIMsgContextFinish

```
axIMsgContextFinish(  
    r_context  
)  
⇒ t
```

Description

Indicates the finish of a message context. Prints any buffered messages in the context and clears the context buffer.

Arguments

r_context Context handle from `axIMsgContextStart`.

Value Returned

t Always returns *t*.

Example

See the “[Message Handler Example](#)” on page 613.

```
context = axIMsgContextStart()  
... do some things ...  
axIMsgContextFinish(context)
```

axlMsgContextClear

```
axlMsgContextClear(  
    r_context  
)  
⇒ t
```

Description

Clears the buffered messages for a context.

Note: You should not normally use this function. Use functions that print (`axlMsgContextPrint`) or retrieve (`axlMsgContextGetString`) messages and then clear them, or use a context finish (`axlMsgContextFinish`) that forces the messages to be printed.

Arguments

<code>r_context</code>	Context handle from <code>axlMsgContextStart</code> . If <code>r_context</code> is nil, clears all messages in all contexts.
------------------------	--

Value Returned

<code>t</code>	Always returns <code>t</code> .
----------------	---------------------------------

Example

See the [“Message Handler Example”](#) on page 613.

```
context = axlMsgContextStart()  
... do some things ...  
axlMsgContextClear(context)
```

axlMsgCancelPrint

```
axlMsgCancelPrint(  
)  
⇒ t
```

Description

Prints a message informing the user that he requested *cancel*. Since the severity of this message is *Error* (3), AXL writes it to the context buffer as it does other warning, error, and fatal messages. Call this function only from a routine that requests user input.

Arguments

None

Value Returned

t	Always returns t.
---	-------------------

Example

See the “[Message Handler Example](#)” on page 613.

```
axlMsgCancelPrint()
```

Sets severity level to 2 (warning) and puts the following into the message buffer (if axlMsgSys messages are in English):

```
User CANCEL received
```

axlMsgCancelSeen

```
axlMsgCancelSeen(  
)  
⇒ t/nil
```

Description

Checks to see if the `axlMsgCancelPrint` message was printed. Does not poll the input stream for a *CANCEL* key. Checks the buffer of current messages for the occurrence of the `axlSsgSys.cancelRequest` message.

Arguments

None

Value Returned

t	Requested <i>cancel</i> has been seen.
nil	No requested <i>cancel</i> has been seen.

Example

See the “[Message Handler Example](#)” on page 613.

```
if( axlMsgCancelSeen()  
    ; clear input and return  
    ...
```

axlMsgClear

```
axlMsgClear()  
)  
⇒ t
```

Description

Clears the current error severity level. You can reset the severity by first saving it using axlMsgTest, then setting it using axlMsgSet.

Arguments

None

Value Returned

t	Always returns t.
---	-------------------

Example

See the [“Message Handler Example”](#) on page 613.

```
axlMsgClear()
```

axIMsgSet

```
axIMsgSet(  
    x_class  
)  
⇒ t
```

Description

Sets the current error severity level. Use only to reset the severity cleared using axIMsgClear.

Arguments

x_class Severity level.

Value Returned

t Always returns t.

Example

See the “[Message Handler Example](#)” on page 613.

```
level = axIMsgTest()  
; Do something  
...  
axIMsgSet(level)
```

axIMsgTest

```
axIMsgTest(  
)  
⇒ x_class
```

Description

Determines the current error severity level. Returns a single number giving the severity level of the most severe message that has been printed since the last `axIMsgClear` or `axIMsgContextClear/Print/GetMsg` call.

Arguments

None.

Value Returned

<i>x_class</i>	Highest severity level of all the messages currently in the message buffer.
----------------	---

Example

See the “[Message Handler Example](#)” on page 613.

```
level = axIMsgTest()  
⇒ 4
```

Design Control Functions

AXL-SKILL Design Control Functions

The chapter describes the AXL-SKILL functions you can use to get the name and type of the current design.

axlCurrentDesign

```
axlCurrentDesign()  
)  
⇒ t_design
```

Description

Returns the name of the active layout which is the name following the label “ALLEGRO LAYOUT:” in the PCB Editor design window title bar.

Arguments

None.

Value Returned

<i>t_design</i>	Always returns the name of the active layout as a string, even if the name is unnamed.brd.
-----------------	--

Example

```
axlCurrentDesign()  
⇒ "mem32meg"
```

Gets the name of the active layout.

axlDesignType

```
axlDesignType(  
    g_detailed  
)  
⇒ t_type
```

Description

Returns a string giving the type of the active design. The level of the type returned depends on the argument *g_detailed*, as described below.

Arguments

g_detailed If *g_detailed* is nil, returns the high-level types "LAYOUT" to denote a layout design, and "SYMBOL" to denote a symbol definition design. If *g_detailed* is t, returns a detailed value describing the design as follows:

For a layout type design: "BOARD", "MCM", "MDD" (module), or "TIL" (tile).

For a symbol type design: "PACKAGE", "MECHANICAL", "FORMAT", "SHAPE", or "FLASH".

Value Returned

t_type One of the string values described.

Example 1

```
axlDesignType(nil)  
⇒ "LAYOUT"
```

Gets the high-level design type of the current active layout.

Example 2

```
axlDesignType(t)  
⇒ "BOARD"
```

Gets the detailed design type.

axlCompileSymbol

```
axlCompileSymbol(  
    ?symbol t_name  
    ?type t_type  
)  
⇒ t_symbolName/nil
```

Description

Compiles and edit checks the current (symbol) design and saves the compiled version on disk under the name *t_name*. If *t_name* is not provided, then *t_name* is the current drawing name. If *t_name* is provided as *nil*, you are prompted for the name. Uses the symbol type in determining some of the edit checks.

Arguments

<i>t_name</i>	Name of the compiled symbol.
<i>t_type</i>	Type of symbol, the default is the current symbol type (see axlDesignType).

Value Returned

<i>t_symbolName</i>	Name of the symbol.
<i>nil</i>	Error due to incorrect arguments.

Note: This function is only available in the symbol editor. This is a SKILL interface to the PCB Editor “create symbol” command and thus has the same behavior.

axlSetSymbolType

```
axlSetSymbolType(  
    t_symbolType  
)  
⇒ t_symbolType/nil
```

Description

Sets the PCB Editor symbol type. Has some minor effect on the commands available and the edit checks performed when the symbol is “compiled” ([axlCompileSymbol](#)).

Arguments

t_symbolType "package", "mechanical", "format", "shape" or
 "flash"

Value Returned

t_symbolType Symbol exists.

nil Error due to incorrect argument.

Note: This function is only available in the symbol editor. This is a SKILL interface into the change symbol type in the Drawing Parameters dialog box in *allegro_symbol*.

axlDBControl

```
axlDBControl(  
    s_name  
    [g_value]  
)  
g_currentValue/ls_names
```

Description

Inquires and/or sets the value of a special database control. If the setting is a value, the return is the old value of the control.

A side effect of most of these controls is that, if an active dialog box is displaying the current setting, the setting may not be updated. Additional side effects of individual controls are listed. Items will be added in the future. Items currently supported for *s_name* include the following:

Name: *drcEnable*

Value: t or nil

Set?: Yes

Description: Enables or Disables DRC/CBD system.

Equivalent: Same as status dialog box drc on/off buttons.

Side Effects: Does not perform batch DRC.

Name: *ratnestDistance*

Value: t or nil

Set?: Yes

Description: Accesses the ratsnest display mode.

t - pin to pin

nil - closest dangling net cline/via

Equivalent: Same as status dialog box Ratsnest Dist control.

Side Effects: Will recalculate the display when this is changed.

Name: *activeTextBlock*

Value: 1 to *maxTextBlock*

Set?: Yes

Description: Accesses the active text block number.

Equivalent: Same as status dialog box text block.

Side Effects: None

Allegro PCB and Package User Guide: SKILL Reference

Design Control Functions

Name: *maxTextBlock*

Value: 16 to 64

Set?: No

Description: Reports the maximum text block number.

Equivalent: None

Side Effects: None

Name: *activeLayer*

Value: <class> / <subclass>

Set?: Yes

Description: Accesses the current class/subclass.

Equivalent: Same as ministatus display.

Side Effects: None

Name: *defaultSymbolHeight*

Value: float

Set?: Yes

Description: Sets the default symbol height in the database.

Equivalent: Status dialog box's DRC Symbol Height.

Side Effects: Will set DRC out of date, and does not update status dialog box if it is present.

Name: *highlightColor*

Value: 1 to 24

Set?: Yes

Description: Queries/changes the permanent highlight color. Can be used with axlHighlightObject

Equivalent: Color form, Display Group, Permanent highlight

Side Effects: Will also change the perm highlight color in the hilite command. Call axlVisibleUpdate to update display.

Name: *tempColor*

Value: 1 to 24

Set?: Yes

Description: Queries/changes the temporary highlight color. Can be used with axlHighlightObject

Equivalent: Color form, Display Group, Temporary highlight

Side Effects: Call axlVisibleUpdate to update display.

Name: *ratsnestColor*

Value: 1 to 24

Set?: Yes

Description: Queries/changes the ratsnest color

Equivalent: Color form, Display Group, Ratsnest Color

Side Effects: Call axlVisibleUpdate to update display.

Allegro PCB and Package User Guide: SKILL Reference

Design Control Functions

Name: *gridColor*

Value: 1 to 24

Set?: Yes

Description: Queries changes the grid color

Equivalent: Color form, Display Group, Grid Color

Side Effects: Call `axlVisibleUpdate` to update display.

Name: *schematicBrand*

Value: none concepthd1, capture, scald

Set?: No

Description: Queries current database schematic branding.

Equivalent: See netin PCB Editor dialog.

Side Effects: None.

Name: *cmgrEnabledFlow*

Value: t or nil

Set?: Yes (only if flag is set)

Description: Reports if board is in constraint manager enabled flow. Only valid in HDL

Concept Flow. t - Cadence schematic cmgr flow enabled (5 pst files required)

nil - traditional Cadence schematic flow (3 pst files)

Equivalent: Import Logic Branding shows "Constraint Manager Enabled Flow"

Side Effects: It is not advisable to clear this flag. This interface is provided for those customers who enabled the flow and want to restore the traditional flow. You need to do additional work on the schematic side to clear the option.

Name: *schematicDir*

Value: directory path

Set?: Yes

Description: Queries/changes the Cadence schematic directory path. This is the directory location for the Cadence `pst` files. This does not support 3rd party netlist location. If database is unbranded (see above) then the directory location is not stored. Existence of location is not verified.

Equivalent: Import Logic Branding Cadence Tab

Side Effects: none

Name: *testPointFixed*

Value: t or nil

Set?: Yes

Description: Sets global flag to lock test points. t - test points fixed nil - test points not fixed

Equivalent: Same as `testprep` param "Fixed test points"

Side Effects: None

Name: *ecsetApplyRipupEtch*

Value: t or nil

Allegro PCB and Package User Guide: SKILL Reference

Design Control Functions

Description: If enabled, and the schedule is anything other than min tree, ripup any etch that disagrees with the schedule. Etch may be ripped up when a refdes is renamed due to this option. Only applicable with Allegro PCB Design 610 tools. t - ripup etch nil - preserve etch

Equivalent: Same as constraint manager *tools->options "Rip up etch..."*

Side Effects: None

Name: *maxEtchLayers*

Value: number

Set?: No

Description: Returns maximum number of etch subclasses.

Equivalent: none

Side Effects: none

Name: *dynamicFillMode*

Value: wysiwyg, rough, nil (Disabled)

Set?: Yes

Description: Controls filling of dynamic shapes.

Equivalent: shape global param

Side Effects: Disabled in symbol editor.

Name: *cnsAreaIgnoreUntype*

Value: t or nil

Set?: Yes

Description: Controls new versus traditional constraint area behavior. In traditional mode (nil) if you do not NET_SPACING_TYPE and /or NET_PHYSICAL_TYPE on areas and nets, then these objects get a default value of 'NO_TYPE'. This causes all constraint areas and objects to participate in both physical and spacing rules for each constraint area.

In new mode (t), you need to add one of the above properties to the objects to have if use the constraint area. With this mode of operation you can create constraint areas that just applies to physical or spacing.

Equivalent: cns master dialog -- checkbox

Side Effects: None

Name: *maxAttachmentSize*

Value: integer value in bytes

Set?: No

Description: Returns the largest attachment size (axlCreateAttachment) that may attach to the database.

Equivalent: None.

Side Effects: None.

Name: *activeLayer*

Name: *newFlashMode*

Value: t/nil

Set?: No

Description: Returns nil if board is running old style or t if board is running new style flash mode (WYSIWYG negative artwork using fsm files).

Note: WYSIWYG is now smooth in the display.

Equivalent: None

Side Effects: N/A

Arguments

s_name Symbol name of control. nil returns all possible names.

g_value Symbol value to set. Usually a t or a nil.

Value Returned

g_currentValue Old value of the control.

ls_names If name is nil, then returns a list of controls.

Example 1

```
old = axlDBControl ('drcEnable, nil)
```

Sets drc system off.

Example 2

```
current = axlDBControl ('ratsnestDistance)
```

Gets current value of pin-2-pin ratsnest.

Example 3

```
listOfNames = axlDBControl (nil)
```

Gets all names supported by this interface.

axlDBSectorSize

```
axlDBSectorSize(  
    [f_size]  
)  
⇒ l_result/nil
```

Description

Retrieves current sector and obstacle values. If *f_size* is specified, this function first changes the database (*x* & *y*) sector size to the specified value, then retrieves new sector and obstacle values.

Arguments

f_size Sector size (user units). If not supplied, size is not changed.

Value Returned

l_result (int) New obstacle count list containing new
 (float) X sector size
 (float) Y sector size
 (int) Number of columns (X)
 (int) Number of rows (Y)
 (int) Original obstacle count
 List containing old
 (float) X sector size
 (float) Y sector size
 (int) Number of columns (X)
 (int) Number of rows (Y)

nil Error due to incorrect argument.

Allegro PCB and Package User Guide: SKILL Reference

Design Control Functions

Note: Sets `x` and `y` sector size to the same value. If `nil`, then only the current counts are returned, and they are in the *new* section of the list.

axlGetDrawingName

```
axlGetDrawingName(  
)  
⇒ t_drawingPathName
```

Description

Retrieves the full path of the drawing.

Arguments

None

Value Returned

t_drawingPathName Full path of the drawing.

axlKillDesign

```
axlKillDesign(  
)  
⇒ t/nil
```

Description

Same as (axlOpenDesign <unnamed> "wf"), where <unnamed> is the standard PCB Editor name provided for an unnamed design. If the specific name is important, you can determine it using the [axlCurrentDesign](#) command.

Arguments

None

Value Returned

t	New design opened, replacing current design.
nil	No new design opened.

axlOpenDesign

```
axlOpenDesign(  
    ?design t_design  
    ?mode t_mode  
)  
⇒ t_design/nil
```

Description

Opens a design. The new design replaces the current design. If the current design has unsaved edits, the user is asked to confirm before discarding them unless the current design was opened in "r" mode (not supported in this release) or unless *g_mode* is "wf". If the user cancels the confirmmer, the function returns nil and the current design remains open.

If *t_design* is not provided, the user is prompted for the name of the design to open.

If *t_design* does not exist on disk, the standard PCB Editor Drawing Parameters dialog boxd appears.

The new design is of the same type as the current design. To reset the design type, specify the appropriate extension with the design name. For example, use a .dra extension to open a file with design type set to SYMBOL, or use a .brd or .mcm extension to open a file with design type set to LAYOUT.

Arguments

t_design Name of the new design to edit

g_mode "w" or "wf" (see above)

Value Returned

t_design New design name.

nil No design opened due to incorrect argument.

Note: Functions the same as the PCB Editor *edit* command, except you can set the *t_mode* to "wf" in order to discard the current edits without confirmation.

Since confirmation uses the standard PCB Editor confirmmer, using the "wf" mode is the same as setting the NOCONFIRM environment variable.

axlRenameDesign

```
axlRenameDesign(  
    t_design  
)  
⇒ t_design/nil
```

Description

Changes the current design name. Has no effect on the existing disk version of the current design. The new current design name will be displayed in the window border and becomes the default name for axlSaveDesign.

Arguments

t_design New current design name.

Value Returned

t_design New current design name.

nil Error due to incorrect argument.

axlSaveDesign

```
axlSaveDesign(  
    ?design t_design  
    ?mode t_option  
)  
⇒ t_design/nil
```

Description

Saves the design with the name specified (*t_design*). If *t_design* is not specified, the current design name is used. If *t_design* is provided but the value is *nil*, the user is prompted for the name. If *t_option* is "nocheck", the database "quick check" is not provided. Use this option only when there is a very compelling reason.

Arguments

<i>t_design</i>	Name for the saved design.
<i>t_option</i>	"nocheck" - No database check performed.

Value Returned

<i>t_design</i>	Name for the saved design.
<i>nil</i>	Error due to incorrect argument(s).

Note: Essentially the PCB Editor *write* command. The current design name is not changed. Use *axlRenameDesign* to change the current design name.

axlDBChangeDesignExtents

```
axlDBChangeDesignExtents(  
    l_bBox  
)  
⇒ t/nil
```

Description

Changes design extents. This may fail if an object falls outside the new extents or if the extents exceed the database range.

Arguments

l_bBox New design extents.

Value Returned

t Size changed.

nil Failed to change size.

Note: This function may take extra time on large designs.

Example 1

```
extents = axlExtentDB('obstacle')
axlDBChangeDesignExtents(extents)
```

Reduces the database to its smaller extent.

Example 2

```
extents = axlExtentDB('obstacle')
extents = bBoxAdd(extents '((-100 -100) (100 100))
axlDBChangeDesignExtents(extents)
```

To reduces the database to its minimum size plus 100 mils all around.

axlDBChangeDesignOrigin

```
axlDBChangeDesignOrigin(  
    l_point  
)  
⇒ t/nil
```

Description

Changes the origin of the design. This may fail if the new design origin falls outside the maximum integer range.

This is an offset. A negative number moves the database left or down and a positive number moves the database to the right or up.

Note: This may take extra time on large designs.

Arguments

l_point X/Y offset.

Value Returned

t Changed the origin of the design.

nil Failed to change the origin of the design due to an incorrect argument.

Example

```
axlDBChangeDesignOrigin(900:900)
```

Moves the origin.

axlDBChangeDesignUnits

```
axlDBChangeDesignUnits(  
    t_units/nil  
    x_accuracy/nil  
    x_drcCount/nil  
)  
⇒ x_drcCount/nil
```

Description

Changes the units and accuracy of the design. To maintain the current design value, use `nil` for `t_units` or `x_accuracy`.

When you change units, also change accuracy to maintain adequate precision within the database. Reference the following table to determine the appropriate accuracy.

Units	Min Accuracy	Max Accuracy	Delta
mils	0	4	0
inches	2	4	3
microns	0	2	0
millimeters	1	4	3
centimeters	2	4	4

- Use the new `DELTA` to decide what the accuracy should be when changing units.

```
new acc = orig acc + (new delta - old delta)
```

- This example shows mils to millimeters with a current accuracy of 1:

```
new acc      = orig acc+ (millim delta- mils delta)  
        4      = 1      +     3      -     0
```

- When changing from one English unit to another or from one Metric unit to another, the default accuracy must match that of the previous unit. For example, if your design uses `MILLS 0` and you change to `INCHES`, the accuracy must be set to 3.
- If you change from an English unit to a Metric unit or from a Metric unit to an English unit, then the accuracy must be set to a number that is at least as accurate as the current database. For example, if the design is in `MILLS 0` and you change to `MILLIMETERS`, then

the accuracy must be set to 2. Then if you change to INCHES, the accuracy must be set to 3 (inches and 3 = mils and 0.)

- If the accuracy needed is not allowed due to a limitation on the number of decimal places allowed for a particular unit, or if you reduce the level of accuracy, the error message, "E-Accuracy will be decreased, database round-offs may occur," displays. For example, if your design is in MICRONS and an accuracy of 1 and you change to CENTIMETERS, you get an error since CENTIMETERS are not allowed with an accuracy of 5.

Arguments

t_units Mils, inches, millimeters, centimeters, or microns.

x_accuracy Range is 0 to 4, according to the table shown.

Value Returned

x_drcCount drc count.

nil Failed to change design units.

Notes:

- This function may take extra time on large designs.
- This function reruns DRC if enabled.

Example 1

```
axlDBChangeDesignUnits("millim" 4)
```

Changes a design from mils/0 to millim/4 (accuracy maintains database precision).

Example 2

```
axlDBChangeDesignUnits(nil 2)
```

Increases accuracy from 0 to 2 and keeps the same units.

axlDBCheck

```
axlDBCheck(  
    g_option/lg_options  
    [p_file]  
)  
⇒ (x_errors x_warnings)/nil
```

Description

Runs *dbdoctor* on the current database. By default, no log file is produced. You can specify the '*log*' option which writes the standard *dbdoctor.log* file. A port descriptor can be the second argument to write the *dbdoctor* output to an external file.

Arguments

g_option/lg_options

Option	Function
'general	Performs a full database check.
'links	Performs a full database check.
'branch	Performs a full database check.
'shapes	Checks shapes (autovoid) <ul style="list-style-type: none">■ May be slow for complicated shapes.■ Normally fast.■ Slow on large databases.
'all	Performs a full check and fix. Does not perform a shape check.
'drc	Performs a batch DRC.
'log	Uses the standard <i>dbdoctor.log</i> file for output.
<i>p_file</i>	Port to write dbcheck logging.

Value Returned

(*x_errors x_warnings*) Results of the check.

Examples

```
res = axlDBCheck ('all)
printf ("errors = %d; warnings = %d", car (res), cadr (res);

p = outfile ("mylogfile")
res = axlDBCheck('(links shapes) p)
close(p)
```

axIDBCopyPadstack

```
axIDBCopyPadstack(  
    rd_dbid  
    l_startEnd  
    [g_dontTrim]  
)  
⇒ o_dbid/nil
```

Description

Creates a new padstack from an existing padstack. The name for the new padstack automatically derives from the existing name by adding a post fix that does not collide with any existing names. The padstack is marked in the database as derived from its starting padstack.

You must provide legal, etch, start and end layers, but *g_dontTrim* must be *t*. If trim is enabled, the new padstack only has pads between the list layers.

Trimming only restricts resulting padstack to between the two indicated layers. If the starting padstack does not already span between the two layers then it will not expand to fill those layers.

Arguments

<i>o_dbid</i>	<i>dbid</i> of the padstack to copy from.
<i>l_startEnd</i>	List of start (<i>class/subclass</i>) and stop (<i>class/subclass</i>) layers.
<i>g_dontTrim</i>	Use <i>t</i> if you do not want the padstack trimmed, or <i>nil</i> to trim the padstack if necessary.

Value Returned

<i>dbid</i>	<i>dbid</i> of the new padstack.
<i>nil</i>	Failed to create new padstack.

Example

- 1) To derive an exact copy:

```
newPadId = axlDBCopyPadstack(padId, '( "ETCH/TOP" "ETCH/BOTTOM" ) )
```

- 2) To derive and trim to only connect from top layer to 2

```
newPadId = axlDBCopyPadstack(padId, '( "ETCH/TOP" "ETCH/2" ) t )
```

axlDBDelLock

```
axlDBDelLock(  
    [t_password]  
)  
⇒ t/nil
```

Description

Deletes a lock on the database. If the database is locked with a password, you must supply the correct password to unlock it.

Arguments

t_password Password string no longer than 20 characters.

Value Returned

t Lock is removed or there is no lock to remove.

nil Failed to remove lock due to an incorrect password.

Example 1

```
axlDBDelLock( )
```

Deletes a lock with no password.

Example 2

```
axlDBDelLock( "mypassword" )
```

Deletes a lock with the password, mypassword.

axlDBGetLock

```
axlDBGetLock(  
)  
⇒ nil/l_info
```

Description

Returns information about a lock on the database. You retrieve the following information:

<i>t_userName</i>	User login who locked the database.
<i>t_lockDate</i>	Date the database was locked.
<i>t_systemName</i>	System on which the database was locked.
<i>t_export</i>	String representing the current setting for enabling/disabling the ability to export design data to other formats.
<i>t_comments</i>	Comments optionally set by the user who locked the database.

You can also determine whether or not a database is locked as this function returns `nil` when the database is unlocked, and returns a list of information when the database is locked.

Arguments

`none`

Value Returned

<code>nil</code>	Database is unlocked.
<code>l_info</code>	List (<i>t_userName</i> <i>t_systemName</i> <i>t_export</i> [<i>t_comments</i>]). Database is locked.

axlDBSetLock

```
axlDBSetLock(  
    [ 'password t_password]  
    [ 'comments t_comments]  
    [ 'exports "disabled"/"enabled" ]  
)  
⇒ t/nil
```

Description

Locks the database against future changes. After setting the lock, you must save the database in order to save the lock. After saving the lock, you can no longer save changes to the database.

User is warned when opening a locked database. Attempts to save the database fail with the exceptions of saving the initial lock and uprev.

Arguments

t_password If provided, you need this to unlock the database. If you forget the password, you cannot unlock the database. The password must be 20 characters or fewer. The following characters are not allowed in the password: (\ whitespace) or leading dash (-)

t_comments Free form comments. You may embed carriage returns (\r) in the string to force a new line in the locking user interface.

'exports Default is to allow exporting data to other formats. If disabled, the following exports are prevented: techfile write, dump_libraries, and create modules.

Note: Upreving the database from one version to the next ignores the lock flag.

Value Returned

t	Locked database.
nil	Failed to lock database.

Example 1

```
axlDBSetLock()
```

Locks the database (basic lock).

Example 2

```
axlDBSetLock('comments "I locked the database"  
             'exports "disabled")  
axlSaveDesign(?design "locked_data")  
printf("Locked info %L\n" axlDBGGetLock())
```

Locks the database, includes comments, and disables data export.

axlPadStackToDisk

```
axlPadStackToDisk(  
    [t_padName]  
    [t_outPadName]  
)  
⇒ t/nil
```

Description

Saves a board padstack out to a library.

Arguments

<i>t_padName</i>	Name of the pad to be saved to a library.
<i>t_outPadName</i>	Name of the output pad.

Value Returned

t	Pad is created.
nil	Failed to create pad.

Example 1

```
axlPadstackToDisk()
```

Dumps all the padstacks in the layout.

Example 2

```
axlPadstackToDisk("pad60cir36d")
```

Dumps padstack "pad60cir36d" from the layout as "pad60cir36d.pad".

Example 3

```
axlPadstackToDisk("pad60cir36d" "mypadstack")
```

Dumps padstack "pad60cir36d" from the layout as "mypadstack.pad".

axlTechnologyType

```
axlTechnologyType(  
)  
⇒ t_technology
```

Description

Returns the type of design technology in use.

Arguments

none

Value Returned

<i>t_technology</i>	Technology type as shown:
---------------------	---------------------------

Technology Type

"mcm"

"pcb"

Product

Allegro Package Designer 610 or Allegro Package SI 610

PCB Editor

axlTriggerClear

```
axlTriggerClear(  
    s_trigger  
    s_function  
)  
⇒ t/nil
```

Description

Removes a registered callback trigger. You pass the same arguments you passed as you registered the trigger.

Arguments

s_trigger Trigger type. See `axlTriggerSet`.

s_function Trigger function to remove - the same as you passed to `axlTriggerSet`.

Value Returned

t Trigger removed.

nil Failed to find a trigger by the specified name.

Example

See `axlTriggerSet` for an example.

axlTriggerPrint

```
axlTriggerPrint()  
)  
⇒ t
```

Description

Debug function that prints what is registered for triggers.

Arguments

none

Value Returned

t	Always returns t.
---	-------------------

axlTriggerSet

```
axlTriggerSet(  
    s_trigger  
    s_function  
)  
⇒ t/nil  
  
axlTriggerSet(  
    nil  
    nil  
)  
⇒ (ls_listOfSupportTriggers)
```

Description

Allows an application to register interest in events that occur in PCB Editor. When called with both arguments as `nil`, returns a list of supported triggers.

Restrictions

Unless otherwise indicated, in your Skill callback trigger you cannot:

- open, save or close the current database.
- call `axlShell`.
- call any of the `axlEnter` functions or any of the `Select`.
- object functions to request a user pick.

You should restrict any user interaction to blocking dialogs (forms).

Notes

- All trigger functions take a single argument. If you provide a function that does not match this standard, your trigger is **not** called.
- Any processing you do in triggers increases the time to open or save a database. Keep it short! If it must be longer, inform the user regarding the processing delay using `axlUIWPrint`.

- You can register multiple functions for a single trigger, but each registration must have a different function. The order that these functions are called when the trigger occurs is undefined.
- PCB Editor sends a close trigger when PCB Editor exits normally. Abnormal exits such as crash, user kill, etc. result in the trigger not being generated.
- You can do user interface work in triggers. You must have the user enter all information before returning from the trigger. Opening a dialog box may involve returning before getting data from the user. To avoid this problem, call `axlUIWBlock` after `axlFormDisplay`. This ensures you do all processing inside the trigger. For correct dialog box display, use the block option in `axlFormCreate`, the `lt_placement` parameter.
- The triggers for opening and saving the database are generally not called when the database does temporary saves, for example, `axlRunBatchDBProgram`, `reports`, or `netrev`.
- You can disable triggering by setting the environment variable `dbg_noskilltriggers`. If you suspect that applications running on triggers are causing problems, set this environment variable to help you debug.

Arguments

<code>s_trigger</code>	Trigger type as shown:
s_trigger Option	Description
<code>'open (t_database g_existing)</code>	Called immediately after a database is opened. Passed a list of two items: the name of the database and <code>t</code> if existing or <code>nil</code> if a new database. Restrictions: You should not open, save, and close the current database.
<code>'save (t_oldName t_newName)</code>	Called before a database is saved to disk. Passed a list of two items: the old name of the database and the new name of the database. If these are the same, then the database was overwritten. This is not called when autosaving.
<code>'close t_name</code>	Called before another database is opened. The single item is the name of the database being discarded.
<code>s_function</code>	Callback function for the event. Each trigger should have its own function. If you use the same function for multiple triggers you can not determine what function caused the trigger.

Value Returned

t	Trigger is registered for the indicated callback.
nil	Trigger is not registered for the indicated callback.

Examples

See `<cdsroot>/share pcb/examples/skill/trigger` for an example.

Example 1

In `ilinit`, add:

```
procedure( MyTriggerOpen( t_open )
    let( (brd old)
        brd = car(t_open)
        old = cadr(t_open)
        if (old then
            old = "Existing"
        else
            old = "New"
        )
        printf("SKILLCB Open %s database %s\n" old brd)
        t
    )))
axlTriggerSet('open 'MyTriggerOpen)
```

Shows how to use this with `~/pcbenv/allegro.ilinit` to be notified when your user opens a new board. Echoes a print every time a user opens a new database.

Example 2

```
( isCallable('axlTriggerSet) axlTriggerSet('open 'MyTriggerOpen))
```

To be compatible with pre-14.1 software, substitute for `axlTriggerSet` in `allegro.ilinit`.

axlGetActiveLayer

```
axlGetActiveLayer()  
)  
⇒ t_layer
```

Description

Retrieves active class and subclass of the design.

Note: This function is obsolete and is kept for compatibility reasons. Use `axlDBControl`.

Arguments

None

Value Returned

t_layer Returns a string denoting the active class and subclass.

Example

```
axlGetActiveLayer()  
⇒ "MANUFACTURING/PHOTOPLOT_OUTLINE"
```

axlGetActiveTextBlock

```
axlGetActiveTextBlock()  
)  
⇒ _textBlock
```

Description

Gets the current active text block, equivalent to the status dialog box.

Arguments

None

Value Returned

_textBlock Returns the current active text block number.

axlSetActiveLayer

```
axlSetActiveLayer(  
    t_layer  
)  
⇒ t/nil
```

Description

Sets the active class and subclass of the design.

Note: This function is obsolete and is kept for compatibility reasons. Use `axlDBControl`.

Arguments

`t_layer` String with the format: `<class>/<subclass>`.

Value Returned

`t` Set the given active class and subclass successfully.

`nil` Failed to set active the given class and subclass.

Example

```
axlSetActiveLayer( "MANUFACTURING/PHOTOPLOT_OUTLINE" )  
⇒ t
```

Allegro PCB and Package User Guide: SKILL Reference

Design Control Functions

Database Group Functions

Overview

This chapter describes the AXL-SKILL Database Group functions.

axlDBAddGroupObjects

```
axlDBAddGroupObjects(  
    o_group  
    lo_members  
)  
⇒ t/nil
```

Description

Adds the database objects specified in the new members list to a group. All restrictions and disclaimers specified in `axlDBCreateGroup` also apply for this procedure.

Arguments

<i>o_group</i>	<i>dbid</i> of the group to receive new members.
<i>lo_members</i>	List of <i>dbid</i> 's specifying the new group members. Database objects already in the group are silently ignored (giving a return value of t.) A single <i>dbid</i> can be substituted for a list.

Value Returned

t	Objects added to the group.
nil	Objects could not be added to the group because the resulting group does not meet the restrictions specified in <code>axlDBCreateGroup</code> .

axlDBCreateGroup

```
axlDBCreateGroup(  
    t_name  
    t_type  
    lo_groupMembers  
)  
⇒ o_dbid/nil
```

Description

Creates a new group database object with members specified by *lo_groupMembers*.

Arguments

t_name String providing the group name. If name is in use by an existing group, this function fails and returns *nil*.

t_type String defining the group type. Legal values include: "generic"

lo_members List of AXL *dbids* defining the group members. Duplicate *dbid* entries are silently ignored. An object is added to a group only once. A single *dbid* can be substituted for a list.

If certain restrictions on the group members are violated, this function fails and returns *nil*. The following are restrictions for all types of groups:

- Only *dbids* of the following *objType* are allowed:

- group
- component
- symbol
- net
- path
- via
- shape
- polygon

pin

text

- Circular group relationships cannot be formed. For example, Group A can not be added as a member of Group B if Group B is directly or indirectly a member of Group A.

Value Returned

o_dbid *dbid* of the newly formed group.

Example

```
groupMembers = axlGetSelSet()  
group_dbid = axlDBCreateGroup("my_group" "generic" groupMembers)
```

Note: The order of the group members provided when you access the *groupMembers* property may vary from the order provided in *lo_groupMembers*.

axlDBDisbandGroup

```
axlDBDisbandGroup(  
    o_group  
)  
⇒ t/nil
```

Description

Disbands the database group you specify with the *o_group* argument, thereby immediately removing the group. Members of the group are not deleted.

Arguments

o_group *dbid* of the group to be deleted.

Value Returned

t Group disbanded.

nil Group could not be disbanded due to an invalid argument, for example, the *dbid* not being for a valid group.

axlDBRemoveGroupObjects

```
axlDBRemoveGroupObjects(  
    o_group  
    lo_members  
)  
⇒ t/nil
```

Description

Removes the database objects from the specified group. Group members, though removed, are not deleted.

Arguments

<i>o_group</i>	Group <i>dbid</i> .
<i>lo_members</i>	List of database objects to be removed from the group. A single <i>dbid</i> can be substituted for a list.

Value Returned

t	One or more objects removed from the group.
nil	<i>lo_members</i> contained no <i>dbids</i> of objects which could be removed from the group.

Notes:

- If a group is left with no members, the group is tagged for deletion, but is not removed immediately.
- You do not need to explicitly remove objects from a group before deleting the object with `axlDeleteObject`. Deleting an object removes it from all groups to which it belongs.

Database Attachment Functions

Overview

This chapter describes the AXL-SKILL Database Attachment functions.

axlCreateAttachment

```
axlCreateAttachment(  
    t_attachmentId  
    t_passwd  
    x_revision  
    s_dataFormat  
    t_data  
)  
⇒ o_attachment/nil
```

Description

Creates a new PCB Editor database attachment with the given attachment id. The attachment may optionally be given a password and a revision number. Attachment data may be specified as a string or as a file name.

axlDBControl('maxAttachmentSize) returns the maximum size of data that can attach to the database.

Arguments

<i>t_attachmentId</i>	Id or name of the attachment to retrieve. Can be up to 31 characters in length.
<i>t_passwd</i>	Password for this attachment. Can be up to 31 characters in length. If no password is desired this may be <i>nil</i> .
<i>x_revision</i>	Revision number of the attachment. If <i>nil</i> , the revision number is set to zero.
<i>s_dataFormat</i>	Indicates the format of the <i>t_data</i> argument. If <i>s_dataFormat</i> is ' <i>string</i> ', the value of the <i>t_data</i> argument is used for the attachment data. If <i>s_dataFormat</i> is ' <i>file</i> ', then the <i>t_dataString</i> argument is interpreted as a file name from which the attachment data is read.
<i>t_data</i>	String of ascii characters representing the attachment data. May represent the data itself or the name of the file from which to read the data, depending on the value of the <i>s_dataFormat</i> argument.

Value Returned

o_attachment AXL id for the new attachment, which can then be queried using the right arrow (->) operator.

nil Failed to create an attachment due to incorrect arguments.

Note: Once an attachment is password protected it needs to be deleted, then re-added to remove or change the password protection.

For additional information, see: `axlIsAttachment`, `axlGetAttachment`,
`axlGetAllAttachmentNames`, `axlSetAttachment`, `axlDeleteAttachment`,
`axlDBControl`

axlDeleteAttachment

```
axlDeleteAttachment(  
    t_attachmentId  
    [t_passwd]  
)  
⇒ t/nil
```

Description

Deletes the given attachment. If the attachment is password protected, the correct password must be given.

Arguments

t_attachmentId Id or name of the attachment to delete.

t_passwd Password for this attachment.

Value Returned

t Attachment deleted.

nil Attachment not deleted.

axlGetAllAttachmentNames

```
axlGetAllAttachmentNames(  
    )
```

⇒ *l_attachment/nil*

Description

Returns a list of the ids for all database attachments in the PCB Editor database. If no attachments are present, then `nil` is returned. You can retrieve the attachments using the `axlGetAttachment()` function.

Arguments

none

Value Returned

l_attachment List of attachment ids.

`nil` No attachments exist in the database.

axlGetAttachment

```
axlGetAttachment(  
    t_attachmentId  
    [s_dataFormat]  
)  
⇒ o_attachment/nil
```

Description

Returns the database attachment with the given id. If the attachment exists, an *attachment record* is returned containing information about the attachment. The data is in the format specified by the *s_dataFormat* argument. If '*file*' format, then the *data* attribute contains a temporary file name to which the data was written. If '*string*', then the *data* attribute contains the attachment data itself. If the *s_dataFormat* argument is omitted or is *nil*, then the *data* attribute is *nil*.

The attachment record has the following attributes:

Name	Type	Set?	Description
<i>objType</i>	string	NO	Is always "attachment".
<i>id</i>	string	NO	Id (name) of the attachment.
<i>password</i>	boolean	NO	<i>t/nil</i> - Indicates if the attachment is password protected.
<i>timeStamp</i>	integer	NO	Indicates the time last modified in seconds.
<i>revision</i>	integer	YES	User defined revision number for the attachment data.
<i>dataFormat</i>	symbol	YES	Indicates the format of the data stored in the "data" attribute and is one of ' <i>file</i> ', ' <i>string</i> ', or <i>nil</i> (in which case the data is not displayed.)
<i>data</i>	string	YES	Attachment data. May be a file name, the data itself, or <i>nil</i> depending on the value of the <i>dataFormat</i> attribute.
<i>timestamp</i>	integer	NO	Indicates the size of the attachment.

Arguments

<i>t_attachmentId</i>	Id or name of the attachment to retrieve. Can be up to 31 characters in length.
<i>s_dataFormat</i>	Format in which the attachment data is stored in the " <i>data</i> " attribute. Must be 'string, 'file, or nil.

Value Returned

<i>o_attachment</i>	AXL id for the attachment structure which can be queried using the right arrow (->) operator.
nil	Attachment does not exist.

Example

```
attachment = axlGetAttachment( "attachmentOne" 'file)
⇒ attachment:attachmentOne
```

axlIsAttachment

```
axlIsAttachment(  
    o_attachment  
)  
⇒ t/nil
```

Description

Determines if the given object is an AXL attachment.

Arguments

<i>o_attachment</i>	Object to check.
---------------------	------------------

Value Returned

t	Object is an attachment.
---	--------------------------

nil	Object is not an attachment.
-----	------------------------------

axlSetAttachment

```
axlSetAttachment(  
    o_attachment  
    [ t_password ]  
)  
⇒ o_attachment/nil
```

Description

Modifies an existing PCB Editor database attachment with the data contained in the given AXL attachment id. Original attachment object must be obtained from the `axlCreateAttachment`, `axlGetAttachment`, or `axlGetAllAttachments` function. The attachment revision number and the attachment data may both be modified.

Format of the data is determined by the `dataFormat` attribute structure, which may be set by the user. If "`dataFormat`" is '`string`', then the value of the `data` attribute is used for the new attachment data. If "`dataFormat`" is '`file`', then the value of the `data` attribute is a file name from which the attachment data is read.

If the existing attachment is password protected, you must provide the correct password or the function fails.

Arguments

<code>o_attachment</code>	AXL id of the existing attachment to be modified. The <code>revision</code> , <code>dataFormat</code> , and <code>data</code> attributes may all be set to new values by the user.
<code>t_password</code>	Password for the given existing attachment. If this does not match the password of the existing attribute, the attachment update fails. If the existing attachment is not password protected, you may omit this.

Value Returned

`o_attachment` Id of the modified attachment.

`nil` Failed to modify the attachment.

Note: Once an attachment is password protected, to remove or change the password protection you must delete and then re-add the attachment.

Allegro PCB and Package User Guide: SKILL Reference
Database Attachment Functions

Database Transaction Functions

Overview

This chapter describes the AXL-SKILL Database Transaction functions.

axlDBCloak

```
axlDBCloak(  
    g_func  
    [g_mode]  
)  
  
⇒ g_return
```

Description

Improves performance and program memory use while modifying many items in the database. You use axlDBCloak to update many etch or package symbols in batch mode. Works like SKILL's eval function. You pass it a function and its arguments using the following format:

```
axlDBCloak ('MyFunc( myargs ) )
```

You can use axlDBCloak to do the following:

- Batch any net based DRC updates.
- Batch connectivity update.
- Optionally batch dynamic shape updates if *g_mode* is 'shape.
- Incorporate an ererset around your function so any SKILL errors thrown are caught by axlDBCloak.

Do not use axlDBCloak in these circumstances:

- If you are adding or deleting non-connectivity database items (for example, loading many lines to a manufacturing layer)
- If you need to interact with the user. Since connectivity is not updated, do not use the axlEnterXXX functions. Instead, get the information from the user first, then do the cloak update.
- If you are reading the database, using cloak does not help and may actually slow performance.
- If making a single change, using Cloak slows performance.

Note: Using Cloak sets any database ids to nil.

The 'shape *g_mode* option improves performance if changes effect any dynamic shapes on the design. Set this if your changes effect ETCH layers.



For effective debugging, first call your function directly from the top level function, then wrap in the cloak call.

Arguments

g_func Function with any of its arguments.

s_mode nil or 'shape

Value Returned

Returns what *g_func* returns.

Example

```
procedure( DeleteSymbols() )
let( (listOfSymbols)

    listOfSymbols = axlDBGetDesign()->symbols
    when( listOfSymbols
        axlDBCloak( 'DeleteDoit(listOfSymbols) 'shape' )
        axlShell("cputime stop"
    )))
procedure( DeleteDoit(listOfDatabaseObjects)

    foreach(c_item listOfDatabaseObjects
        printf("REFDES %s\n", c_item->refdes)
        axlDeleteObject(c_item)
    )
    nil
)
```

Deletes all placed symbols in the database.

axlDBTransactionCommit

```
axlDBTransactionCommit(  
    x_mark  
)  
⇒ t/nil
```

Description

Commits a database transaction from the last transaction mark.

Arguments

<i>x_mark</i>	Database transaction mark returned from axlDBTransactionStart.
---------------	---

Value Returned

t	Database transaction committed.
---	---------------------------------

nil	Database transaction not committed.
-----	-------------------------------------

Example

See axlDBTransactionStart() for an example.

axlDBTransactionMark

```
axlDBTransactionMark(  
    x_mark  
)  
⇒ t/nil
```

Description

Writes a mark in the database that you can use with `axlDBTransactionOops` to rollback database changes to this mark.

When a transaction mark is committed or rolled back, all `axlDBTransactionMarks` associated with that mark are discarded.

Arguments

<code>x_mark</code>	Database transaction mark returned from <code>axlDBTransactionStart</code> .
---------------------	---

Value Returned

<code>t</code>	Mark written in the database.
<code>nil</code>	No mark written in the database.

Example

See `axlDBTransactionStart()` for an example.

axlDBTransactionOops

```
axlDBTransactionOops(  
    x_mark  
)  
⇒ t/nil
```

Description

Undoes a transaction back to the last mark, or to start if there are no marks. Supports the Allegro *oops* model for database transactions.

When a transaction mark is committed or rolled back all, then that mark is no longer valid for *oopsing*.

Arguments

<i>x_mark</i>	Database transaction mark returned from axlDBTransactionStart.
---------------	---

Value Returned

t	Transaction undo completed.
nil	Transaction is already back to the starting mark and there is nothing left to <i>oops</i> .

Example

See axlDBTransactionStart for an example.

axlDBTransactionRollback

```
axlDBTransactionRollback(  
    x_mark  
)  
⇒ t/nil
```

Description

Undo function for a database transaction.

Arguments

<i>x_mark</i>	Database transaction mark returned from axlDBTransactionStart.
---------------	---

Value Returned

t	Transaction undo completed.
---	-----------------------------

nil	Transaction undo not completed.
-----	---------------------------------

Example

See axlDBTransactionStart for an example.

ax1DBTransactionStart

```
ax1DBTransactionStart(  
)  
⇒ x_mark/nil
```

Description

Marks the start of a transaction to the database. Returns a mark to the caller which is passed back to commit, mark, oops or rollback for nested transactions. Only the outermost caller of this function (the first caller) has control to commit or rollback the entire transaction.

You use this function with other `ax1DBTransaction` functions.

Allegro cancels any transactions left active when your SKILL code terminates. You cannot start a transaction and keep it active across Allegro commands as an attempt to support *undo*.

Saving or opening a database cancels transactions.

Arguments

none

Value Returned

<i>x_mark</i>	Integer mark indicating transaction start.
---------------	--

<i>nil</i>	Failed to mark transaction start.
------------	-----------------------------------

Example 1

```
mark = axlDBTransactionStart()
...#1 do stuff ...
axlDBTransactionMark(mark)
...#2 do stuff ...
axlDBTransactionMark(mark)
...#3 do stuff ...

;; do an oops of the last two changes
axlDBTransactionOops( mark ) ; oops out #3
axlDBTransactionOops( mark ) ; oops out #2
axlDBTransactionOops( topList); commit only #1
```

Emulates the Allegro *oops* model.

Example 2

```
i = axlDBTransactionStart()
... do stuff ...
j = axlDBTransactionStart()
... stuff ...
axlDBTransactionCommit(j) ;; this is not really committed

j = axlDBTransactionStart()
... do more stuff ...
axlDBTransactionRollback(j) ;; oops out "do more stuff"

axlDBTransactionCommit(i) ;; commit changes to database
```

Multiple Start marks.

Note: Database transaction functions do NOT mark select sets. The application handles select set management.

Allegro PCB and Package User Guide: SKILL Reference
Database Transaction Functions

Constraint Management Functions

Overview

This chapter describes the AXL-SKILL functions related to constraint management.

For a list of constraints, see Appendix B in the *Allegro Constraint Manager User Guide*.

axlCnsAddVia

```
axlCnsAddVia(  
    t_csetName  
    t_padstackName  
)  
==> t/nil
```

Description

Adds padstack to the constraint via list of a physical cset. Padstack does not need to exist to be added to a constraint via list.

If *t_csetName* is *nil*, add padstack to all physical csets.

Note: If a via already exists in the via list, a *t* is returned. Locked csets return a *nil*.

Arguments

<i>t_csetName</i>	Name of physical cset or <i>nil</i> for all csets.
<i>t_padstack</i>	Name of a via padstack.

Value Returned

<i>t</i>	If added.
<i>nil</i>	Error in arguments; cset does not exist or illegal padstack name.

Examples

Add ALLPAD to all csets

```
axlCnsAddVia(nil "ALLPAD")
```

Add ONEPAD to DEFAULT cset

```
axlCnsAddVia("DEFAULT" "ONEPAD")
```

axlCnsAssignPurge

```
axlCnsAssignPurge(  
    s_tableType  
) ==> x_delCount/ nil
```

Description

Purges either the physical or spacing assignment table of unused entries. PCB Editor supports two assignment tables: physical and spacing. This functionality duplicates that found in the Constraint Assignment Tables forms.

Arguments

s_tableType: Spacing or Physical.

Value Returned

nil Error.

x_delCount Number of entries deleted.

Example

```
axlCnsAssignPurge( 'spacing)
```

See also [axlCnsList](#).

axlCnsDeleteVia

```
axlCnsDeleteVia(  
    t_csetName  
    t_padstackName  
)  
=> t/nil
```

Description

Deletes padstack from the physical via constraint list, *t_csetName*. If *t_csetName* is nil, delete provided padstack from all physical constraint sets.

Notes:

- Will return *t* if asked to delete a via that does not exist in the via list.
- Locked csets will return a *nil*.

Arguments

t_csetName Name of physical cset or nil for all csets.

t_padstack Name of a via padstack.

Value Returned

t If deleted.

nil Error in arguments; cset does not exist or illegal padstack name.

Example

Delete via to default cset

```
axlCnsDeleteVia("DEFAULT" "VIA")
```

Delete via to all csets

```
axlCnsDeleteVia(nil "VIA")
```

See also `axlCnsGetViaList` and `axlPurgePadstacks`.

axlCNSDesignModeGet

```
axlCNSDesignModeGet(  
    nil  
)  
⇒ ls_constraints  
axlCNSDesignModeGet(  
    'all  
)  
⇒ lls_constraintNModes  
axlCNSDesignModeGet(  
    'editable  
)  
⇒ t/nil  
axlCNSDesignModeGet(  
    s_name/t_name  
)  
⇒ s_mode/nil
```

Description

Gets the current DRC modes for checks that fall into the set of design constraints. These constraints pertain to the entire board.

To determine the design constraint checks currently supported, use the following command:

```
axlCNSDesignModeGet()
```

Note: Available constraint checks may change from release to release.

Arguments

nil	Returns all checks in design type DRC
'all	Returns all checks and current mode
'editable	Returns t if mode can be changed, nil mode is not changed and when in PCB Editor studio which does not offer this option
<i>s_name</i>	Symbol name of check.
<i>t_name</i>	String name of check.

Value Returned

<i>ls_names</i>	List of checks (<i>s_name</i> ...)
<i>lls_names</i>	List of checks and their mode ((<i>s_name s_mode</i>) ...)
<i>s_mode</i>	Mode 'on, 'off or 'batch

Example 1

```
axlCNSDesignModeGet(nil)
```

Gets a current list of design constraints.

Example 2

```
axlCNSDesignModeGet('all')
```

Gets a list of settings for all design constraints.

Example 3

```
axlCNSDesignModeGet('Package_to_Package_Spacing')
```

Gets current setting of package to package.

Example 4

```
axlCNSDesignModeGet("Negative_Plane_Islands")
```

Gets current setting of negative plane islands using a string.

axlCNSDesignModeSet

```
axlCNSDesignModeSet(
    t_name/s_name
    t_mode/s_mode
)
⇒ t/nil

axlCNSDesignModeSet(
    'all
    t_mode/smode
)
⇒ t/nil

axlCNSDesignModeSet(
    l_constraintNModes
    t_mode/smode
)
⇒ t/nil

axlCNSDesignModeSet(
    ll_constraintNModes
)
⇒ t/nil
```

Description

Sets the current DRC modes for design constraints. The modes control the DRC for that design constraint check on the entire board.

To determine the checks that are supported, use the following command:

```
axlCNSDesignModeGet()
```

You can set all checks using the argument '`'all`', set individual checks using `t_name`, or set a list of checks to the same mode as follows:

```
'(s_name...) t_mode/s_mode
'(t_name...) t_mode/s_mode
```

You can list sets of checks as follows:

```
'((s_name/t_name s_mode/t_mode) ...)
```

For performance reasons, changing modes or values does not invoke DRC. You must manually invoke DRC. You can mark changes in order to perform fewer DRC updates, depending on your changes (see [axlCNSMapUpdate](#) on page 730.)

Note: Available constraint checks may change from release to release.

Arguments

<i>s_name</i>	Symbol name of check.
<i>t_name</i>	String name of check.
<i>s_mode</i>	Mode setting may be 'on, 'off, or 'batch.
<i>t_mode</i>	String mode setting may be "on", "off" or "batch"
'all	Returns all checks for a given tier of PCB Editor.

Value Returned

<i>t</i>	Success
nil	Failure.

Example 1

```
axlCNSDesignModeSet( 'Package_to_Place_Keepin_Spacing 'on)
```

Turns on package to package keepin check.

Example 2

```
axlCNSDesignModeSet( 'all 'batch)
```

Makes all design constraints batch only.

Example 3

```
axlCNSDesignModeSet( '(Negative_Plane_Islands Pad_Soldermask_Alignment)' off)
```

Turns two constraints off.

Example 4

```
axlCNSDesignModeSet( '( (Package_to_Place_Keepout_Spacing 'on) ) )
```

Sets various constraints to different modes.

For a programming example, see `cns-design.il`, which you can find in the following location:

```
<cdsroot>/share pcb/examples/skill/cmds
```

axlCNSDesignValueCheck

```
axlCNSDesignValueCheck(  
    s_name/t_name  
    g_value  
)  
⇒ (t_string/nil, nil/t_errorMsg)/nil
```

Description

Checks the syntax of the given value against the allowed syntax for the given constraint. You use the function `axlCNSDesignGetValue(nil)` to get the constraint names.

Note: Allowed syntax may change from release to release.

Arguments

<i>s_name</i>	Symbol name of the constraint.
<i>t_name</i>	String name of the constraint.
<i>g_value</i>	Value to verify

Value Returned

(<i>t_string/nil</i>)	Value correct. <i>t_string</i> shows current user unit preference. For example, if you supply "10", the return might be "10.0 MILS" if MILS is the current database unit.
(nil/ <i>t_errorMsg</i>)	Value incorrect. <i>t_errorMsg</i> reflects the error.
nil	Arguments are incorrect.

Examples

```
axlCNSDesignValueCheck( 'Negative_Plane_Islands "10 mils" )
```

Tests if allowed to set.

axlCNSDesignValueGet

```
axlCNSDesignValueGet(  
    nil  
    [g_returnNameString]  
)  
⇒ ls_constraints  
axlCNSDesignValueGet(  
    'all  
    [g_returnString]  
)  
⇒ lls_constraintNValues  
axlCNSDesignValueGet(  
    s_name  
    [g_returnString]  
)  
⇒ f_value/t_value/nil
```

Description

Fetches the values from those design constraints that support values. Use `axlCNSDesignValueGet(nil)` to determine the set of these constraints.

Note: Constraint checks may change from release to release.

Arguments

<code>nil</code>	Returns all checks that support values.
<code>'all</code>	Returns all checks with values and current value.
<code><i>s_name</i></code>	Symbol name of value.
<code><i>t_name</i></code>	String name of value.
<code><i>g_returnNameString</i></code>	Returns constraint names as strings (default is symbol return.)
<code><i>g_returnString</i></code>	By default, this returns native type in user units (a float) for all checks supported. If <code>t</code> , return is a MKS string where <code>nil</code> returns native.

Value Returned

<code>ls_names</code>	List of all controls that support values (symbol.)
<code>lls_constraintNValues</code>	List of all controls with their values <code>'((s_name f_value/t_value) ...)</code> <code>f_value</code> = user unit value, and <code>t_value</code> = MKS string value.

Example 1

```
axlCNSDesignValueGet(nil)
```

Gets a list of design constraints that support values.

Example 2

```
axlCNSDesignValueGet('all 't)
```

Gets a list of settings for all design constraints with values returned as MKS strings.

Example 3

```
axlCNSDesignValueGet('Negative_Plane_Islands)  
= 10.0
```

Gets the current setting of `Negative_Plane_Islands` in user units.

Example 4

```
axlCNSDesignValueGet("Pad_Soldermask_Alignment" t)  
= "10 mils"
```

Gets the current setting of `Pad_Soldermask_Alignment` as a MKS string (this passes in inquiry as a string).

axlCNSDesignValueSet

```
axlCNSDesignValueSet(  
    t_name/s_name  
    f_value/t_value  
)  
⇒ t/nil  
  
axlCNSDesignValueSet(  
    l1_constraintNValues  
)  
⇒ t/nil
```

Description

This sets the value of the design constraint.

To determine the list of supported values, use the following command:

```
axlCNSDesignValueGet(nil)
```

You may set single values or a list of values:

```
'((s_name/t_name f_value/t_value) ...)
```

For performance reasons, changing a value does not invoke DRC. You must manually invoke DRC. See `axlCNSMapUpdate` for a set of interfaces you can use to mark changes in order to perform fewer DRC updates.

Note: Constraint checks may change from release to release.

Arguments

<i>s_name</i>	Symbol name of check.
<i>t_name</i>	String name of check.
<i>f_value</i>	Floating point value provided is assumed to be in the default user unit for the constraint. Value may be rounded.
<i>t_value</i>	If given as a string with MKS type, the value is converted to current user units for the constraint. Rounding may result.

Value Returned

t	Design constraint value set.
nil	Failed to set design constraint value.

Example 1

```
axlCNSDesignValueSet( 'Negative_Plane_Islands 10.0 )
```

Sets a negative plan tolerance to 10 in current database units.

Example 2

```
axlCNSDesignValueSet( 'Negative_Plane_Islands "10.0 mils" )
```

Sets a negative plan tolerance to 10 mils.

Example 3

```
axlCNSDesignValueSet( '((Negative_Plane_Islands "20 inches")  
  (Pad_Soldermask_to_Pad_Soldermask_Spacing 15.9)) )
```

Sets various constraints to different values.

For a programming example, see `cns-design.il` which you can find in the following location:

```
<cdsroot>/share pcb/examples/skill/cmds
```

axlCNSEcsetCreate

```
axlCNSEcsetCreate(  
    t_name  
    [ t_copyName / o_dbidCopyEcset ]  
)  
⇒ o_dbidEcset/nil
```

Description

Creates a new ECset. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets. The name must be legal and less than the maximum length allowed. Function fails if the ECset already exists.

By default, the ECset is created empty. You can provide a second argument to copy the contents of another ECset into the new ECset.

Arguments

<i>t_name</i>	Name of new ECset.
<i>t_copyName</i>	Optional name to copy from.

Value Returned

<i>o_dbidEcset</i>	<i>dbid</i> of the new ECset
<i>nil</i>	Failed due to one of the following: the name is illegal, or the ECset already exists.

Example 1

```
axlCNSEcsetCreate( "MyEmptyEcset" )
```

Creates a new empty ECset.

Example 2

```
p = car(axlDBGetDesign()->ecsets)  
axlCNSEcsetCreate( "MyNewEcset" p)
```

Copies the contents of the first ECset in a list.

axlCNSEcsetDelete

```
axlCNSEcsetDelete(  
    t_name/o_dbidEcset  
)  
⇒ t/nil
```

Description

Deletes an ECset from the PCB Editor database and also deletes the *ELECTRICAL_CONSTRAINT_SET* property from any nets assigned this ECset value. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

If the the ECset is locked, you must unlock it before you can delete it.

Arguments

<i>t_name</i>	ECset name
<i>o_dbidEcset</i>	ECset <i>dbid</i>

Value Returned

<i>t</i>	ECset successfully deleted.
<i>nil</i>	ECset is not deleted because of one of the following: the name is incorrect, or ECset is locked.

Example 1

```
axlCNSEcsetDelete( "UPREV_DEFAULT" )
```

Deletes an ECset by name.

Example 2

```
p = car(axlDBGetDesign()->ecsets)  
axlCNSEcsetDelete(p)
```

Deletes the first ECset in a list of ECsets.

axlCNSEcsetGet

```
axlCNSEcsetGet(  
    t_name  
)  
⇒ o_dbidEcset/nil
```

Description

Returns the *dbid* of the ECset when you request it by the ECset name. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Arguments

t_name ECset name.

Value Returned

o_dbidEcset *dbid* of the ECset requested.

nil Function failed due to an incorrect name.

Examples

```
axlCNSEcsetGet( "MyEcset" )
```

Tests for the existence of an ECset named MyEcset.

axlCNSEcsetModeGet

```
axlCNSEcsetModeGet(  
    nil  
)  
⇒ ls_constraints  
axlCNSEcsetModeGet(  
    'all  
)  
⇒ lls_constraintNModes  
axlCNSEcsetModeGet(  
    s_name/t_name  
)  
⇒ s_mode/nil
```

Description

Returns the current DRC modes for checks that are members of electrical constraints. These modes pertain to the entire board. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Note: Not all checks are available in all levels of PCB Editor. To determine the set of checks supported, use the command: `axlCNSEcsetModeGet()`. Constraint checks may change from release to release.

Arguments

<code>nil</code>	Returns all checks in design type DRC.
<code>'all</code>	Returns all checks and current mode.
<code>s_name</code>	Symbol name of the check.
<code>t_name</code>	String name of the check.

Value Returned

<code>ls_names</code>	List of checks (<i>s_name</i> ...).
<code>lls_names</code>	List of checks and related modes ((<i>s_name s_mode</i>) ...)
<code>s_mode</code>	Returns mode 'on, 'off, or 'batch

Example 1

```
axlCNSEcsetModeGet(nil)
```

Lists currently available electrical constraints.

Example 2

```
axlCNSEcsetModeGet('all')
```

Lists settings for all electrical constraints.

Example 3

```
axlCNSEcsetModeGet('Maximum_Stub_Length')
```

Shows current setting of stub length.

Example 4

```
axlCNSEcsetModeGet("Maximum_Via_Count")
```

Shows current setting of via count.

axlCNSEcsetModeSet

```
axlCNSEcsetModeSet(  
    t_name/s_name  
    t_mode/s_mode  
)  
⇒ t/nil  
  
axlCNSEcsetModeSet(  
    'all  
    t_mode/s_mode  
)  
⇒ t/nil  
  
axlCNSEcsetModeSet(  
    l_constraintNModes  
    t_mode/s_mode  
)  
⇒ t/nil  
  
axlCNSEcsetModeSet(  
    ll_constraintNModes  
)  
⇒ t/nil
```

Description

Sets the DRC modes for checks that are members of the electrical constraints set. These modes control the entire board. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Note: Not all checks are available in all levels of PCB Editor. To determine the set of checks supported, use the command: `axlCNSEcsetModeGet()`. Constraint checks may change from release to release.

You can set all checks using the argument '`'all`', set individual checks using `t_name`, or set a list of checks with the same mode as shown:

```
'(s_name ...) t_mode/s_mode  
'(t_name ...) t_mode/s_mode
```

You can list sets of checks as shown:

```
'((t_name t_mode) ...)  
'((s_name s_mode) ...)
```

For performance reasons, changing modes or values does not invoke DRC. You must manually invoke DRC. See [axlCNSMapUpdate](#) on page 730 for a set of interfaces you can use to mark changes in order to perform fewer DRC updates.

Arguments

<i>s_name</i>	Symbol name of the check.
<i>t_name</i>	String name of the check.
<i>s_mode</i>	Mode setting; may be 'on', 'off', or 'batch'.
<i>t_mode</i>	String mode setting; may be "on", "off", or "batch".
'all	Set all checks for a given tier of PCB Editor.

Value Returned

<i>t</i>	DRC mode set.
nil	DRC mode not set.

Example 1

```
axlCNSEcsetModeSet('Maximum_Via_Count 'off)
```

Turns off max via check.

Example 2

```
axlCNSEcsetModeSet('all 'batch)
```

Makes all electrical constraints batch only.

Example 3

```
axlCNSEcsetModeSet('(Maximum_Crosstalk Route_Delay) 'off)
```

Turns two constraints off.

Example 4

```
axlCNSEcsetModeSet( '((Maximum_Crosstalk off)
    (Route_Delay 'on) (Impedance 'batch)) )
```

Sets various constraints to different modes.

axlCNSEcsetValueCheck

```
axlCNSEcsetValueCheck(  
    s_name/t_name  
    g_value  
)  
⇒ (t/t_errorMsg)/nil
```

Description

Checks the syntax of the given value against the allowed syntax for the given constraint. You use the function `axlCNSEcsetGetValue(nil)` to get the constraint names. Electrical Constraint Set (ECSet) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Note: Allowed syntax may change from release to release.

Arguments

<code>s_name</code>	Symbol name of constraint.
<code>t_name</code>	String name of constraint.
<code>g_value</code>	Value to verify.

Value Returned

<code>t</code>	Syntax is correct.
<code>t_errorMsg</code>	Syntax is incorrect. The message indicates the reason.
<code>nil</code>	Constraint name is not supported.

Examples

```
axlCNSEcsetValueCheck( 'Net_Schedule_Topo "STAR" )
```

Tests if allowed to set.

axlCNSEcsetValueGet

```
axlCNSEcsetValueGet(  
    nil  
    [g_returnNameString]  
)  
⇒ ls_constraints  
  
axlCNSEcsetValueGet(  
    'all  
    [g_returnString]  
)  
⇒ lls_constraintNValues  
  
axlCNSEcsetValueGet(  
    o_ecsetDbid/t_ecsetName  
    s_name  
    [g_returnString]  
)  
⇒ f_value/t_value/nil
```

Description

Fetches the constraint values for a given ECset. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Use `axlCNSEcsetValueGet(nil)` to determine the set of allowable constraints.

Each ECset may have all or none of the allowed constraints.

You can retrieve the ECset values by the ECset name or by its *dbid*. You can get the *dbid* of an ECset by using one of the following commands:

- `axlDBGetDesign() ->ecsets`
- `axlCNSEcsetCreate()`

Note: Constraint checks may change from release to release. Not all checks are available in all levels of PCB Editor.

Arguments

<i>o_ecsetDbid</i>	ECset <i>dbid</i> .
<i>t_ecsetName</i>	ECset name.
<i>nil</i>	Returns all checks that support values.
'all	Returns all checks with values and current value.
<i>s_name</i>	Symbol name of value.
<i>t_name</i>	String name of value.
<i>g_returnNameString</i>	Returns constraint names as strings (default is symbol return)
<i>g_returnString</i>	Default is to return native type for all checks supported, this is in user units (a float). If <i>t</i> , return is an MKS string where <i>nil</i> returns native.

Value Returned

<i>ls_names</i>	List of all controls that support values (symbol).
<i>lls_constraintNValues</i>	List of all controls with their values as shown: '((<i>s_name f_value/t_value</i>) ... <i>f_value</i> = user unit value and <i>t_value</i> = MKS string value.

Example 1

```
axlCNSEcsetValueGet(nil)
```

Gets a current list of design constraints that support values.

Example 2

```
ecsets = axlDBGetDesign()->ecsets
ecset = car(ecsets)
axlCNSEcsetValueGet(ecset 'all t)
```

Gets a list of settings for all design constraints with values returned as MKS strings.

Example 3

```
axlCNSEcsetValueGet( "UPREVED_DEFAULT"  'Maximum_Via_Count)  
= 10.0
```

Gets the current setting of Maximum_Via_Count on ECset UPREVED_DEFAULT.

Example 4

```
axlCNSEcsetValueGet( "UPREVED_DEFAULT"  "Pad_Soldermask_Alignment"  t)  
= "10 mils"
```

Gets the current setting of Pad_Soldermask_Alignment as a MKS string (this passes in inquiry as a string).

axlNetEcsetValueGet

```
axlNetEcsetValueGet(  
    o_itemDbid/t_netName  
    t_cnsName/s_name  
)  
==> t_cnsValue/nil
```

Description

Returns the value of a specific electrical constraint that has been assigned to a given net. Both fixed and user defined constraints may be accessed. This will not return a "flattened" net view of constraints applied to pinpairs. Use `axlCnsNetFlattened` to obtain this constraint view.



If requesting multiple constraints from the same net it is faster to get the dbid of the net and pass that as first argument instead of using the net name.

Arguments

<code>o_itemDbid</code>	dbid of any item that is assigned to a net or Xnet.
<code>t_cnsName</code>	Property name for the constraint to be fetched. This can be either a fixed constraint or a user-defined constraint.
<code>s_name</code>	Symbol name of DRC check (values returned by <code>axlCNSEcsetModeGet(nil)</code>). These names may not exactly match the property name. They do not exist for user-defined properties in the ECset.

Value Returned

<code>t_cnsValue</code>	Value returned as a string.
<code>nil</code>	No value defined for the net.

See also `axlCnsNetFlattened`.

Examples:

Net is part of an ECset (electrical constraint set) which has a MAX_EXPOSED_LENGTH constraint:

```
net = axlSelectByName( "NET" "NET2" )
rule = axlNetEcsetValueGet(net "MAX_EXPOSED_LENGTH" )
```

Net has an override constraint for MAX_VIA_COUNT:

```
rule = axlNetEcsetValueGet( "NET2" "MAX_VIA_COUNT" )
```

Same as above example but uses the DRC check name:

```
rule = axlNetEcsetValueGet( "NET2" 'Maximum_Via_Count )
```

axlCNSEcsetValueSet

```
axlCNSEcsetValueSet(
    o_ecsetDbid/t_ecsetName
    t_name/s_name
    f_value
)
⇒ t/nil

axlCNSEcsetValueSet(
    o_ecsetDbid/t_ecsetName
    ll_constraintNValues
)
⇒ t/nil
```

Description

Sets the value of the ECset DRC. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

To determine the list of supported values, use the following command:

```
axlCNSEcsetValueGet(nil)
```

You may set single values or a list of values. *ll_constraintNValues* represents a list of values as shown:

```
'((s_name/t_name f_value/t_value) ...)
```

Passing a `nil` or empty string " " as a value deletes the constraint from the ECset.

For performance reasons, changing a value does not invoke DRC. You must manually invoke DRC. See [axlCNSMapUpdate](#) on page 730 for a set of interfaces that you use in order to mark changes to perform fewer DRC updates.

Note: Constraint checks may change from release to release.

Arguments

<i>o_ecsetDbid</i>	<i>dbid</i> of the ECset.
<i>t_ecsetName</i>	Name of the ECset.
<i>s_name</i>	Symbol name of constraint.

<i>t_name</i>	String name of constraint.
<i>f_value</i>	Floating point value provided is assumed to be in the default user unit for the constraint. Value may be rounded.
<i>t_value</i>	If given as a string with MKS type, the value is converted to current user units for the constraint. Rounding may result.

Value Returned

<i>t</i>	Set value of ECset DRC.
<i>nil</i>	Failed to set value of ECset DRC due to incorrect argument(s).

Examples

Sets impedance:

```
axlCNSEcsetValueSet( "UPREVED_DEFAULT"
                      'Impedance ALL:ALL:100.0:2)
```

Sets multi-value:

```
axlCNSEcsetValueSet( "UPREVED_DEFAULT"
                      '((Impedance "ALL:ALL:100.0:2") (Maximum_Via_Count 5)))
```

axlCnsGetViaList

```
axlCnsGetViaList()
  t_csetName
)
==>lt_padstacks/nil
```

Description

Returns padstacks defined in a physical constraint set. If the cset name is provided then returns only vias assigned for that cset. Otherwise the function returns vias for all csets. The same vias may appear more than once when using the `nil` option. The order of vias has no meaning and returned padstack names may not exist ("*" indicators in cns physical set dialog).

Arguments

`t_csetName` Name of physical cset.

`nil` Process all csets.

Value Returned

`lt_padstacks` List of padstacks defined in a cset or all csets.

`nil` If no padstacks found or cset not found.

Examples

Report vias in default physical constraint set

```
axlCnsGetViaList( "DEFAULT" )
```

Report vias in all physical constraint sets

```
axlCnsGetViaList( nil )
```

axlGetAllViaList

```
axlGetAllViaList (
    [g_attrVias]
)
==> lo_padstack_dbid
```

Description

Returns a list of all padstacks included in via lists in the design. This is a compilation of all via lists from all constraint sets. Optionally it provides padstacks from net VIA_LIST properties.

The order of padstack dbids depends on the order of constraint sets, VIA_LIST properties and the associated via lists.



This interface will result in the via padstacks being loaded into the design if they are not already loaded.

Arguments

[g_attrVias]	Optional argument to add padstacks that are not included in constraint sets but are provided in some net VIA_LIST attributes.
--------------	---

Value Returned

lo_padstack_dbid	List of padstack dbids .
------------------	--------------------------

nil	The design has empty via lists.
-----	---------------------------------

axlLayerSet

```
axlLayerSet(  
    o_dbid  
)  
==>o_dbid/nil
```

Description

Updates changes to layer parameters. You can only update the color and visibility attributes of a parameter. This is a wrapper for axlSetParam. After completing color or visibility changes, call axlVisibleUpdate to update the display.

Arguments

o_dbid Layer parameter dbid.

Value Returned

o_dbid Layer parameter dbid.

nil If error.

See also `axlSetParam` and `axlLayerGet`.

Examples

Change color of top etch layer:

```
q = axlLayerGet( "ETCH/TOP" )  
q->color = 7  
q->visibility = nil  
axlLayerSet(q)
```

If setting multiple layer colors or visibility, only call visible update after last change:

```
axlVisibleUpdate(t)
```

axlCnsList

```
axlCnsList(  
            s_tableType  
            ) ==> lt_list
```

Description

Returns list of physical or spacing constraint sets. See `axlDBGetDesign() ->ecsets` for list of electrical csets. See also `axlPurgePadstacks`, `axlCnsAssignPurge`, `axlCnsDeleteVia`, `axlCnsAddVia`, and `axlCnsGetViaList`.

Arguments

<i>s_tableType</i>	Spacing or physical.
--------------------	----------------------

axlCNSMapClear

```
axlCNSMapClear(  
)  
⇒ t
```

Description

See [axlCNSMapUpdate](#).

Arguments

none

Value Returned

t Always returns t.

Examples

See [axlCNSMapUpdate](#) on page 730 for an example.

axlCNSMapUpdate

```
axlCNSMapUpdate(  
)  
⇒ x_drcCount/nil
```

Description

This function and `axlCNSMapClear`, which do not support nesting, batch and tune DRC updates from constraint changes made by `axlCNS<xxx>` functions. No `axlCNS<xxx>` functions perform a DRC update. Rather, they set the DRC system out-of-date.

You can run DRC system once on a *set* of constraint changes, which is more efficient than running it as part of each change. You may notice the increased efficiency on large boards.

Arguments

none

Value Returned

<code>nil</code>	There is no matching <code>axlCNSMapClear</code> .
<code>x_drcCount</code>	Number of DRCs caused by batch changes.

Example 1

```
axlCNSMapClear()
axlCNSEcsetModeSet('Maximum_Via_Count 'off)
axlCNSDesignModeSet('all 'on)
axlCNSDesignValueSet('Negative_Plane_Islands 10.0)
axlCNSMapUpdate()
```

Turns off electrical max via check by turning all design checks on and setting the island tolerance to 10.

Example 2

```
axlCNSMapClear()
axlCNSEcsetModeSet('Maximum_Via_Count 'on)
x1CNSMapUpdate()
```

Does one change.

axlCnsNetFlattened

```
axlCnsNetFlattened(  
    o_netDbid/t_netName  
    t_cnsName/s_name  
)  
==> t_cnsValue/nil
```

Description

Permits a view of constraints where explicit pinpair rules are promoted to the net. The information reported by the function is the same as in `show element` under the Properties attached to net heading. It is also in a format used by the third party netlist (`netin`) and in the `pstxnet.dat` file used by `netrev`.

If pinpairs are constrained by an electrical rule (for example, `PROPAGATION_DELAY`), PCB Editor stores the constraints on the pinpair, not on the net. The electrical constraints stored on the net are those applied to dynamic pinpairs (the use of the `AD:AR`, `L:S`, syntax) or where the rule applies to the net (for example, `MAX_VIAS`).

This does not return all constraint values applied to the net, if the constraint is obtained via the electrical constraint set (ECset) or overrides exist at the bus or diffpair level. This information is reported in `show element` under the heading, Electrical constraints assigned to net. PCB Editor maps electrical constraints from xnets, matched groups, and pin pairs to nets by promoting or flattening the electrical property to present a traditional net view of the constraints and to provide compatibility with schematic netlisters. Additional constraints may effect the net because of the ECset assigned to the net, xnet, differential pair or bus level. Additional override properties may exist at the differential pair or bus level. You can use `axlNetECsetValueGet`, but it will not flatten constraints.



Tip
When requesting multiple constraints from the same net, use the dbid of the net as first argument instead of the net name.

Arguments

o_netDbid/t_netName dbid or name (string) of the net.

t_cnsName Property name for the constraint.

s_name Symbol name of DRC check (values returned by `axlCNSECsetModeGet(nil)`). These names may not exactly match the property name.

Value Returned

t_cnsValue Value returned as a string exactall.

nil No value defined for the net.

Examples

Get impedance rule by name on `NET1`:

```
rule = axlCnsNetFlattened( "NET1"  "IMPEDANCE_RULE" )
```

Get impedance rule by DRC check name on `NET1`:

```
rule = axlCnsNetFlattened( "NET1"  'Impedance )
```

Get `PROPAGATION_DELAY` on `MEM_DATA8` using the `dbid` of net:

```
net = car(axlSelectByName( "NET"  "MEM_DATA8" ))  
rule = axlCnsNetFlattened(net "PROPAGATION_DELAY" )
```

See also `axlNetEcsetValueGet`.

Allegro PCB and Package User Guide: SKILL Reference
Constraint Management Functions

Command Control Functions

Overview

The chapter describes the AXL-SKILL functions that register and unregister AXL-SKILL functions with PCB Editor and set various modes in the user interface.

AXL-SKILL Command Control Functions

This section lists the command control functions.

axlCmdRegister

```
axlCmdRegister(  
    t_allegroCmd  
    ts_callback  
    ?cmdType t_cmdType  
    ?doneCmd ts_doneCmd  
    ?cancelCmd ts_cancelCmd  
)  
⇒ t/nil
```

Description

Registers a command named *t_allegroCmd* with the PCB Editor shell system. If the command already exists, either because it is a base PCB Editor command or because it has been registered by `axlCmdRegister`, SKILL-AXL executes the more recently registered command. It *hides* the original command.

Once you register a command, PCB Editor passes its arguments to SKILL-AXL without parsing them.

You can call `axlCmdRegister` any time. Once you have registered a command, you can type it at the PCB Editor command line and incorporate it into menus and pop-ups.

Arguments

t_allegroCmd Name of the command to register. Use lowercase letters for the command name.

ts_callback Name of SKILL callback routine called when the command is activated from the PCB Editor window.

t_cmdType String denoting the type of this command:
"interactive"

PCB Editor interactive command which is the default.

"general"

Immediate command that executes as soon as the command is called, even during another command. Use for display refresh commands, for example.

"sub_cmd"

Must be called inside an interactive command. Use for pop-ups.

ts_doneCmd

Done callback function that PCB Editor calls when the user types *done* or selects *Done* from the pop-up. Can be either a symbol or string. If *nil*, PCB Editor calls *axlFinishEnterFun* by default.

ts_cancelCmd

Cancel callback function that PCB Editor calls when the user types *cancel* or selects *Cancel* from the pop-up. This argument can be either a symbol or a string. If it is *nil*, PCB Editor calls *axlCancelEnterFun* by default.

Value Returned

t Command registered successfully.

nil Command not registered.

Example 1

```
axlCmdRegister( "my swap gates" 'axlMySwapGates  
?cmdType "interactive" ?doneCmd 'axlMySwapDone  
?cancelCmd 'axlMySwapCancel)  
⇒ t
```

Registers the command `my swap gates` as calling the function `axlMySwapGates`.

Example 2 SKILL Function Example

For commands (`s_allegroCmd` value) that accept parameters as strings, the SKILL function converts the parameters to symbols.

```
axlCmdRegister( "do it" 'do_print)  
do it myFile Text AshFindAllText
```

Sample `axlCmdRegister` with a registered function that takes arguments where `myFile` is an output port.

```
(defun do_print (arg1 description find_func) ;; Added to deal with strings  
    myport = evalstring(arg1) ;; Added to deal with strings  
    do_print2( myport description find_func) ;; Added to deal with strings  
        ) ; do_print ;; Added to deal with strings  
# Here is the real function  
(defun do_print2 (p description find_func)  
    (let (list)  
        (fprintf p "Properties on %s:\n" description)  
        (setq list (apply find_func nil))  
        (print_list_props list p)  
        (fprintf p "\n\n")  
        ) ; let  
    ) ; do_print2
```

Parse and process the `do it` arguments.

axlCmdUnregister

```
axlCmdUnregister(  
    t_allegroCmd  
)  
⇒ t/nil
```

Description

Unregisters or removes from the PCB Editor shell system, a previously registered command named `t_allegroCmd`. If the command already exists because it is a base PCB Editor command, the original command is available again.

Arguments

`t_allegroCmd` Name of command to be unregistered.

Value Returned

`t` Command unregistered successfully.

`nil` Failed to unregister the specified command.

Example

```
axlCmdUnregister( "my swap gates" )  
⇒ t
```

Unregisters the command `my swap gates`.

axlEndSkillMode

```
axlEndSkillMode(  
)  
⇒ t
```

Description

Returns from the SKILL command mode to the program's command line. The SKILL exit function is mapped to this function. In a SKILL program this command has no effect.

Arguments

None

Value Returned

t	Always returns t.
---	-------------------

Example

```
axlEndSkillMode()  
⇒ t
```

Exits AXL-SKILL.

axlFlushDisplay

```
axlFlushDisplay(  
)  
⇒ t
```

Description

Flushes all data from the display buffer to the display screen itself. Displays items intended to be displayed, but not yet displayed because no event has triggered a flush of the display buffer.

You can display the following items:

- Visible objects added to the database
- Messages
- Highlighting and dehighlighting of selected objects
- Pending display repairs

Generally, AXL delays screen updates until a prompt for user input occurs or an AXL program completes, such as `axlEnterPoint`. Certain programs, such as those that spend long times doing calculations between screen updates, might want to call `axlFlushDisplay` after each batch of screen updates to indicate the progress of the command. Overuse of this call may hurt performance.

Arguments

None

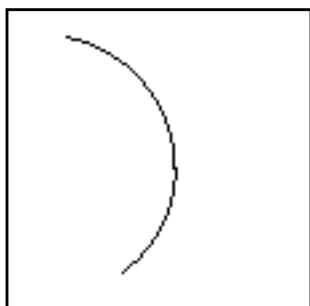
Value Returned

t Always returns t.

Example

```
mypath = axlPathStart( list(8900:4400))
axlPathArcRadius(mypath, 12., 8700:5300, nil, nil, 500)
myline = axlDBCreatePath( mypath, "etch/top" nil)
; Arc is not yet visible
axlFlushDisplay()
; Now arc is visible
```

Creates a path and displays it immediately regardless of whether the user has caused a display event such as moving the cursor into the PCB Editor window.



axlOKToProceed

```
axlOKToProceed()  
)  
⇒ t
```

Description

Checks whether PCB Editor is processing another interactive command or engaged in some process that might interfere with a SKILL command. Use this to check before starting functions such as `dbcreate`, user interactions, and select set operations. Returns `t` if PCB Editor is ready to properly execute a SKILL command, and returns `nil` if it is not.

Arguments

<code>t</code>	Suppresses the error message produced when it is not OK to proceed.
----------------	---

Value Returned

<code>t</code>	PCB Editor can allow AXL-SKILL database, selection, and interactive functions to execute.
----------------	---

<code>nil</code>	PCB Editor cannot allow AXL-SKILL database, selection, and interactive functions to execute.
------------------	--

Example

```
(when axlOKToProceed  
    ;; do AXL interactive command  
)
```

axlSetLineLock

```
axlSetLineLock(
    ?arcEnable      g_arcEnable
    ?lockAngle      f_lockAngle
    ?minRadius      f_minRadius
    ?length45       f_length45
    ?fixed45        g_fixed45
    ?lengthRadius   f_lengthRadius
    ?fixedRadius    g_fixedRadius
    ?lockTangent    g_lockTangent
)
⇒ t/nil
```

Description

Sets one or more of the line lock parameters. The parameters are the same as those accessible in the *Line Lock* section of the PCB Editor Status form.

All parameters not explicitly set in a call to `axlSetLineLock` keep their current settings.

Arguments

`g_arcEnable` If `t`, sets Lock Mode to Arc. Otherwise Lock Mode is Line.
Default is Line.

`f_lockAngle` Sets Lock Direction. Allowed values are: 45 (degrees), 90 (degrees), or 0 (off, or no lock).

`f_minRadius` Sets Minimum Radius, a value in user units.

`f_length45` Sets the Fixed 45 Length value in user units.

`g_fixed45` If `t`, sets the Fixed 45 Length mode. You cannot set this parameter unless `f_lockAngle` is 45, and `g_arcEnable` is nil.

`f_lengthRadius` Sets Fixed Radius value in user units.

`g_fixedRadius` If `t`, sets the Fixed Radius mode. You cannot set this parameter unless `f_lockAngle` is 45 or 90, and `g_arcEnable` is `t`.

`g_lockTangent` If `t`, sets the Tangent mode to on.

Value Returned

- | | |
|-----|--|
| t | Set the given line lock parameters successfully. |
| nil | Failed to set the given line lock parameters. |

Example

```
axlSetLineLock( ?arcEnable t ?lockAngle 90 ?fixedRadius t  
?lengthRadius 50)
```

Sets the Line Lock parameters to Lock Direction: 90, Lock Mode: Arc, Fixed Radius: on at 30 mils.

axlSetRotateIncrement

```
axlSetRotateIncrement(  
    ?angular  f_angular  
    ?radial   f_radial  
)  
⇒ t/nil
```

Description

Sets the dynamic rotate angle increment in degrees (*f_angular*) or radians (*f_radial*). Sets the rotate increment for rotation of objects in the dynamic buffer.

Arguments

<i>f_angular</i>	Sets angle lock increment in degrees.
<i>f_radial</i>	Sets radial lock increment in radians.

Value Returned

t	Set the given rotate increment parameters successfully.
---	---

Example

```
(axlSetRotateIncrement ?angular 15)  
⇒ t
```

Sets the dynamic rotate angle to 15 degrees.

axlUIGetUserData

```
axlUIGetUserData(  
)  
⇒ r(userData/nil
```

Description

Gets the current user data structure from PCB Editor. The user data structure stores basic information about the state of the user interface. By default it contains the properties:

<i>doneState</i>	How the user returned control to the application. Possible values are either: <i>done</i> or <i>cancel</i> .
<i>popupId</i>	List of the current pop-up values. A list of string pairs, as shown: (("Done" "axlFinishEnterFun") ("Cancel" "axlCancelEnterFun"))
<i>ministatForm</i>	Always <i>nil</i> . (Reserved for future releases.)

You can set your own attributes in the user data structure to communicate with callbacks, as shown in the example. You cannot overwrite the three basic attributes: *doneState*, *popupId*, or *ministatForm*.

Arguments

None

Value Returned

<i>r(userData</i>	User data structure from PCB Editor.
<i>nil</i>	Failed to get user data structure from PCB Editor.

Example

```
userdata = axlUIGetUserData()  
userdata->??  
⇒ (doneState cancel  
popupId (( "Cancel" "axlCancelEnterFun" ))  
ministatForm nil  
)
```

axlUIPopupDefine

```
axlUIPopupDefine(  
    r_popup  
    ts_pairs  
)  
⇒ r_popup/nil
```

Description

Creates a pop-up from the name value pair list *ts_pairs*. If *r_popup* already exists, it appends the name value pairs at the end of the existing pairs in *r_popup*. Use the returned *r_popup* id as the argument to `axlUIPopupSet` to make it the active pop-up.

Arguments

<i>r_popup</i>	Predefined pop-up handle to which to append new entries. Can be <code>nil</code> to create a new pop-up.
<i>ts_pairs</i>	List containing the pairs ((<i>t_display t_callback</i>) defining each pop-up entry display name and its AXL function callback.

Value Returned

<i>r_popup</i>	Id of the pop-up created or updated.
<code>nil</code>	No pop-up created or updated.

Example

```
popid = axlUIPopupDefine( nil
    (list (list "Complete" 'axlMyComplete)
          (list "Rotate" 'axlMyRotate)
          (list "Something" 'axlMySomething)
          (list "Cancel" 'axlCancelEnterFun))

⇒ (      ("Complete" 'axlMyComplete)
          ("Rotate" 'axlMyRotate)
          ("Something" 'axlMySomething)
          ("Cancel" 'axlCancelEnterFun))
```

Creates a pop-up with the following selections:

- *Complete*, associated with your function `axlMyComplete`
- *Rotate*, associated with `axlMyRotate`
- *Something*, associated with `axlMySomething`
- *Cancel*, associated with `axlCancelEnterFun`

axlUIPopupSet

```
axlUIPopupSet(  
    r_popup  
)  
⇒ t/nil
```

Description

Sets the active pop-up in PCB Editor. If *r_popup* is `nil`, unsets the currently active pop-up.

You can clear the active pop-up by calling outside of the form callback, as follows:

```
axlUIPopup (nil)
```

Arguments

r_popup Predefined pop-up handle created by `axlUIPopupDefine`.

Value Returned

`t` Set *r_popup* as the active pop-up.

`nil` Failed to set *r_popup* as the active pop-up.

Example

```
popid = axlUIPopupDefine( nil
    (list (list "Complete" 'axlMyComplete)
          (list "Rotate" 'axlMyRotate)
          (list "Something" 'axlMySomething)
          (list "Cancel" 'axlCancelEnterFun))

⇒ (      ("Complete" 'axlMyComplete)
          ("Rotate" 'axlMyRotate)
          ("Something" 'axlMySomething)
          ("Cancel" 'axlCancelEnterFun))
```

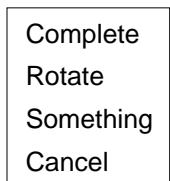
Creates a pop-up with the following selections:

- *Complete*, associated with your function `axlMyComplete`
- *Rotate*, associated with `axlMyRotate`
- *Something*, associated with `axlMySomething`
- *Cancel*, associated with `axlCancelEnterFun`

```
axlUIPopupSet( popid)
    ⇒ t
```

Sets up the pop-up.

You can then move the cursor into the PCB Editor window and press the middle mouse button. PCB Editor displays the following pop-up:



If, for example, you select *Complete*, PCB Editor performs the callback to `axlMyComplete`.

axlBuildClassPopup

```
axlBuildClassPopup(  
    r_form  
    t_field  
)  
⇒ t/nil
```

Description

Supports building a form pop-up with a list of classes.

Arguments

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name in form or pop-up name of form.

Value Returned

t	Pop-up built.
nil	Failed to build pop-up due to incorrect arguments.

Examples

```
axlBuildClassPopup(fw, "CLASS")  
axlFormSetField(fw, "CLASS" axlMapClassName( "ETCH" ))
```

axlBuildSubclassPopup

```
axlBuildSubclassPopup(  
    r_form  
    t_field  
    t_class  
)  
⇒ t/nil
```

Description

Supports building a form pop-up with a list of subclasses from the indicated class.

Arguments

<i>r_form</i>	Form handle
<i>t_field</i>	Field name
<i>t_class</i>	Class name

Value Returned

t	Built form subclass pop-up.
nil	Failed to build form subclass pop-up due to incorrect arguments.

Example

```
# place holder since popup will be overridden by the code
POPUP <subclass>"subclass" "subclass".
...
# field name should match t_field
FIELD subclass
FLOC 9 3
ENUMSET 19
OPTIONS prettyprint ownerdrawn
POP "subclass"
ENDFIELD
axlBuildSubclassPopup(fw, "subclass" axlMapClassName( "ETCH" ))
axlFormSetField(fw, "subclass" "GND")
```

Form file entry provides a subclass pop-up with color swatch support.

Note: axlMapClassName supports Allegro Package Designer 610 and Allegro Package SI 610, which rename certain classes.

axlSubclassFormPopup

```
axlSubclassFormPopup(  
    r_form  
    t_field  
    t_class  
    nil/lt_subclass  
)  
⇒ t/nil
```

Description

Builds a form pop-up for a given PCB Editor class for a given field of *r_form* using the `axlSubclassFormPopup` function. This function is a combination of `axlGetParam` and `axlFormBuildPopup` with color swatching.

If the fourth argument is `nil`, the pop-up is based on all subclasses of the given class.

You can easily build a subclass pop-up containing current colors as swatches. To do this, add the following in the form file for that field:

```
OPTIONS ownerdrawn
```

Pop-ups built this way are dispatched back to the application as strings.

Note: if list of subclasses are passed, illegal subclass names are silently ignored.



To take advantage of color swatches in your subclass pulldown (ENUMSET) use this interface and the `ownerdrawn` option in the form file. The form file entry for your control should look like:

```
FIELD <field name>  
  
FLOC <x y location>  
  
ENUMSET <width of field>  
  
OPTIONS prettyprint ownerdrawn  
  
POP <popup name>  
  
ENDFIELD
```

Note *prettyprint* option upper/lower cases the popup name the user sees.

Arguments

<i>r_form</i>	Standard form handle (see axlFormCreate on page 502)
<i>t_field</i>	Field name in form or pop-up name of form.
<i>t_class</i>	Class name.
<i>nil/lt_subclass</i>	Use <code>nil</code> for all members of the class, otherwise specify a list.

Value Returned

- | | |
|-----|------------------------------|
| t | Form pop-up built. |
| nil | Failed to build form pop-up. |

Example

```
axlSubclassFormPopup( form "subclass_name" "ETCH" nil)
```

axlVisibleUpdate

```
axlVisibleUpdate(  
    t_now  
)  
⇒ t
```

Description

The `axlVisible` family and its base building block permit changing layer color and visibility.

```
    axlSetParam( "paramLayerGroup: . . . " )
```

You can also use these functions in conjunction with Find Filter interaction to permit filtering objects by layer via changing visibility.

The SKILL application must indicate display update via `axlVisibleUpdate` when changing visibility on the user.

Updates any forms that display color or visibility to the user.

For most situations, pass `nil` to this function. This defers updating the main graphics canvas until control is returned to the user, allowing a combination of several canvas updates into one update.

Arguments

<code>t</code>	Update now.
<code>nil</code>	Update when control is returned to the user.

Value Returned

<code>t</code>	Returns <code>t</code> always.
----------------	--------------------------------

Example 1

```
; ; You should not interact with the user when you have  
; ; visibility modified  
; ; get current visibility  
p = axlVisibleGet()  
axlVisibleDesign(nil) ; turn off all layers  
; ;... change visibility of selected layers ...  
; ;... Selection of objects without user interaction  
; restore visibility  
axlVisibleSet(p)
```

Selects items using visibility without updating the display.

Example 2

```
; ; only leave top etch layer on  
axlVisibleLayer("etch" nil)  
axlVisibleLayer("etch/top" t)  
; update display when control is returned to the user  
axlVisibleUpdate(nil)
```

Updates the display after changing visibility.

Example 3

```
p = axlGetLayer("etch/top")  
  
; legal numbers are 1 to 24  
p->color = 10  
axlSetParam(p)  
  
;make this call after you change all colors and visibility  
axlVisibleUpdate(t)
```

Changes color on top etch layer.

axlWindowFit

```
axlWindowFit(  
)  
⇒ l_bBox
```

Description

Zooms in to (or out of) a design fitting it fully on the window. For the PCB Editor in layout mode, performs a fit on the outline. For the PCB Editor symbol mode, performs a fit such that all visible objects occupy maximum window area. Returns the bounding box of the window after the fit has been performed.

Arguments

none

Value Returned

l_bBox	The bounding box of the window after zooming (in user units).
--------	---

Note: This is available as the PCB Editor command *window fit*.

Polygon Operation Functions

Overview

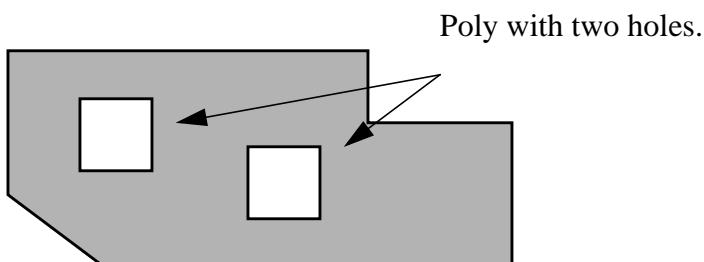
This chapter describes the AXL/SKILL Polygon Operation functions and includes the following sections:

- About Polygon Operations
- AXL-SKILL Polygon Operation Attributes
- AXL-SKILL Polygon Operation Functions
- Use Models

About Polygon Operations

A *poly* is a set of points linked so that the start and end point are the same. A poly is always non-intersecting and may contain *holes*. A hole is a non-intersecting closed loop enclosed within a poly.

Figure 20-1 Poly with Holes



Note: These polys refer to the *o_polygon* object in AXL-SKILL. Polys are different from the AXL Skill Database *polygon* object which represents an PCB Editor unfilled shape.

Allegro PCB and Package User Guide: SKILL Reference

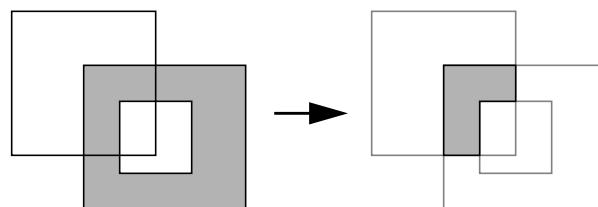
Polygon Operation Functions

These functions, which perform geometric operations on polys, are called logical operations and do the following:

- Create route keepouts based on the board outline, offset by a pre-determined distance.
- Create split planes from route-keepin and Anti-etch.
- Automatically generate a package keepout surrounding a package and its pins, contoured around the package.

Logical operations in AXL-SKILL enable SKILL programmers to do the following:

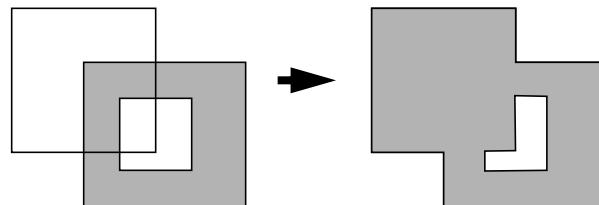
- Create logical operation objects (*lo_polygon*) from these PCB Editor database objects:
 - pins
 - lines
 - clines
 - vias
 - shapes
 - rectangles
 - frectangles (filled rectangles)
 - voids
- Create *o_polygon* from holes in a poly (*o_polygon*)
- Create an PCB Editor database shape from an *o_polygon*
- Perform geometric operations on *lo_polygons*, resulting in the creation of other *lo_polygons*.
 - Logical AND - The intersection of two polys.



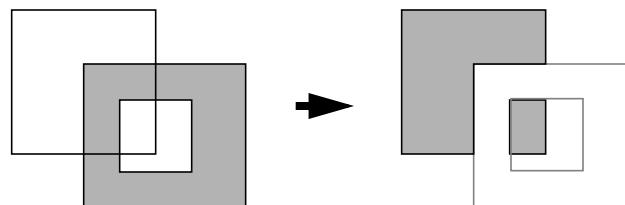
Allegro PCB and Package User Guide: SKILL Reference

Polygon Operation Functions

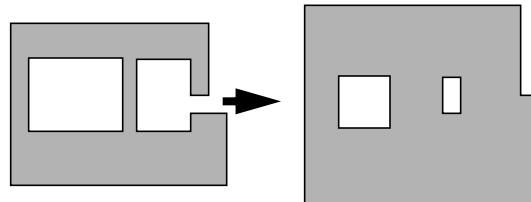
- ❑ Logical OR - The union of two polys.



- ❑ Logical ANDNOT - Subtracting one poly from another.



- ❑ Logical EXPAND (CONTRACT is expand in the opposite direction.)



- Perform query operations on a *o_polygon*, including the following:
 - ❑ The area of the poly
 - ❑ The bounding box around the poly
 - ❑ The holes and vertices of a poly
 - ❑ If the *o_polygon* is a hole
- Access path data and holes from an *o_polygon*
- Locate a point respective to the poly

Error Handling

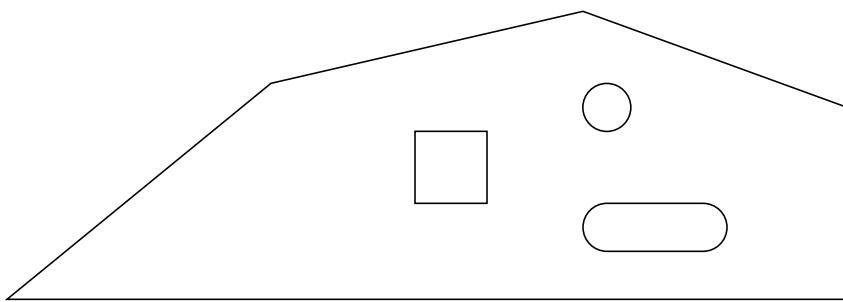
Return values for each function are specified along with the description of the function, in case of error.

AXL-SKILL Polygon Operation Attributes

The following are the attributes of the *o_polygon* type:

Attribute Name	Type	Description
<i>area</i>	float	Area of the poly in the same units as the drawing.
<i>bBox</i>	bBox	Poly's bounding box.
<i>vertices</i>	list	Path-like data of the outer boundary of the poly available as a list of lists where each of the sublists will contain a point representing a vertex of the poly and a floating point number representing radius of the edge from the previous vertex to the present vertex. No arcs spanning across quadrants or greater than 90 degrees.
		Radius of 0.0 indicates a straight line edge between the two vertices.
		Positive radius value indicates that the arc lines to the left of the center and negative radius imply that the arc lies to the right of the center of the arc.
<i>holes</i>	list	<i>lo_polygon</i>
<i>isHole</i>	boolean	<i>t = hole, nil = poly</i>

The following figures illustrate the attributes described.



■ Attributes

```
vertices (((9775.0 775.0) 0.0)
          ((9100.0 1075.0) 0.0)
          ((8350.0 950.0) 0.0))
holes (poly:20199696 poly:20199896
       poly:22204476)
isHole nil
```

Note: All the 3 polys representing holes of the above poly have their isHole attribute set to t.



■ Attributes

```
vertices (((5975.0 976.0) 0.0)
          ((5787.0 788.0) 188.0)
          ((7225.0 599.0) 0.0)
          ((7413.0 787.0) -188.0)
          ((7225.0 975.0) -188.0))
holes nil
isHole nil
```

AXL-SKILL Polygon Operation Functions

This section lists the polygon operation functions.

axlPolyFromDB

```
axlPolyFromDB(  
    o_dbid  
    ?endCapType      s_endCapType  
    ?layer          t_layer  
    ?padType        s_padType  
)  
⇒ lo_polygon/nil
```

Description

Creates a list of *o_polygon* objects from the *dbid*. Use the *lo_polygon* list to get the poly attributes or to perform logical operations on these polys.

Arguments

o_dbid PCB Editor *dbid* of one of the following database types from which to construct the poly.

line
cline
shape
rect
frect
pin
via
void

r_path Path construct from the axlPath API family. This is not an PCB Editor database object so it is a much more efficient method than creating an PCB Editor shape; then converting it to a Poly. Note axlDBCreatOpenShape also supports an *r_path*.

Allegro PCB and Package User Guide: SKILL Reference

Polygon Operation Functions

<i>s_endCapType</i>	Keyword string specifying the end cap type to use for the poly, one of 'SQUARE, 'OCTAGON, or 'ROUND. Used in case of line or cline only, otherwise ignored. Default is 'SQUARE.
<i>t_layer</i>	Keyword string specifying the layer of the pad to retrieve, for example, "ETCH/TOP". Used in the case of pin or via only, otherwise ignored. Default is "ETCH/TOP".
<i>s_padType</i>	Keyword string specifying the type of the pad to be retrieved, one of 'REGULAR, 'ANTI, or 'THERMAL. Used in the case of pin or via only, otherwise ignored. Default is 'REGULAR.

Value Returned

<i>lo_polygon</i>	Object representing the resulting geometry.
<i>nil</i>	Cannot get polys.

Example

```
polyList = axlPolyFromDB(via_dbid, ?layer "ETCH/BOTTOM" ?padType 'ANTI')
```

axlPolyMemUse

```
axlPolyMemUse (
) -> lx_polyCounts
```

Description

This returns a list of integers reflecting the internal memory use of the axlPoly interfaces. If you assign Poly objects to global handles (instead of assigning to locals, e.g let or prog statements) then you need to insure all of global data is nil-ed at the end of your program. The example below shows how to check that you have written your program correctly.

Description of 5 integers. Integers 2 through 5 are for Cadence use.

- 1 - Most important and shows number of Skill Polys still in use.
- 2 - Number of Allegro Polys in use. This is always \geq to Skill Polys. The additional polys are voids (holes) in the Skill polys.
- 3 - Number of edges in all Allegro polys.
- 4 - Number of Allegro Floating Point Polys (should be 0).
- 5 - Number of edges in all Allegro Floating Point Polys (should be 0).

Arguments

None

Value Returned

lx_polyCounts A list of 5 integers reflecting Poly memory usage.

See also

[axlPolyOperation](#)

Example

Verify at end of your program you have no hanging Poly memory in use.

```
gc() ; requires Skill development licenses
```

Allegro PCB and Package User Guide: SKILL Reference

Polygon Operation Functions

```
a xlPolyMemUse( )  
;; should return all 0's
```

axlPolyOffset

```
axlPolyOffset (
    o_polygon/lo_polygon
    l_xy
    [g_copy]
)
=> o_polygon
```

Description

This offsets the entire poly by the provided xy coordinate. Optionally if g_copy is t it will copy the poly, default is to offset the provided poly.

Note: The offseted polygon must be entirely within the extents of the drawing.

Arguments

<i>o_polygon</i>	<i>o_polygon</i> on which the operation is to be done.
<i>lo_polygon</i>	Optionally pass a list of polys.
<i>l_xy</i>	Coordinates in user units for offset.
<i>g_copy</i>	Optional, if t does the offset on a copy.

Value Returned

<i>lo_polygon/</i>	
<i>o_polygon</i>	In place offset (<i>g_copy nil</i>) or offseted copy of polygond (<i>g_copy</i> is t). If passed a list of polys returns a list otherwise return a poly.

See also

[axlPolyFromDB](#)

Example

See the following.

```
<cdsroot>/share pcb/examples/skill/axlcore/ashpoly.il
```

axlPolyOperation

```
axlPolyOperation
  o_polygon1 / lo_polygon1
  o_polygon2 / lo_polygon2

  s_operation
)
⇒ lo_polygon/nil
```

Description

Performs the logical operation specified on the two sets of polys. Does not allow hole polys as input. When holes are passed as input, the following warning is displayed:

Invalid polygon id argument -<argument>



Underlying polygon operation function fails and returns nil in rare dense geometrical situations.

Arguments

o_polygon1 / lo_polygon1 *o_polygon* or list of *o_polygons* on which the operation is to be done.

o_polygon2 / lo_polygon2 List of *o_polygons* on which the operation is to be done.

s_operation String specifying the type of logical operation, one of 'AND, 'OR, or 'ANDNOT.

Value Returned

lo_polygon List of *o_polygons* which represent the geometry resulting from performing the operation on the arguments.

nil Error due to incorrect arguments.

To be more specific:

(*o_polygon_out1 o_polygon_out2 ...*) is returned if the result after performing the operation is a list of polys.

`nil` is returned if the result after performing the operation is a `nil` poly. For example, consider performing the `AND` operation on two non-overlapping sets of polys.

`nil` is returned if the operation fails. You can obtain a descriptive error message by calling `axlPolyErrorGet()`.

Example

```
poly1_list = (axlPolyFromDB cline dbid)
    poly2_list = (axlPolyFromDB shape_dbid)
    res_list = (axlPolyOperation poly1_list poly2_list 'OR)
```

axlPolyExpand

```
axlPolyExpand(  
    o_polygon1 / lo_polygon1  
    f_expandValue  
    s_expandType  
)  
⇒ lo_polygon/nil
```

Description

This function yields a list of polys after expanding them by a specified distance. Use of a negative number causes contraction. Distance is specified in user units. This function does not allow hole polys as input. When holes are passed as input, the following warning is displayed:

Invalid polygon id argument -<argument>



Underlying polygon operation function fails and returns nil in rare dense geometrical situations.

Arguments

o_polygon1 /lo_polygon1

o_polygon / list of *o_polygons* on which the operation is to be done.

f_expandValue Amount of expansion in user units.

s_expandType Symbol specifying the exterior corners of the geometry during expansion, one of:

- 'NONE - no corner modifications
- 'ALL_ARC
- 'ACU_BLUNT
- 'ACU_ARC

Allegro PCB and Package User Guide: SKILL Reference

Polygon Operation Functions

Value Returned

<i>lo_polygon</i>	List of <i>o_polygons</i> which represent the resulting geometry after performing the expansion on the specified polys.
nil	Failed to expand polys due to incorrect arguments.

To be more specific:

(*o_polygon_out1 o_polygon_out2 ...*) is returned if the result after performing the operation is a list of polys.

nil is returned if the result after performing the operation is a *nil* poly, for example, consider contracting a 20x30 rectangle by 40 units.

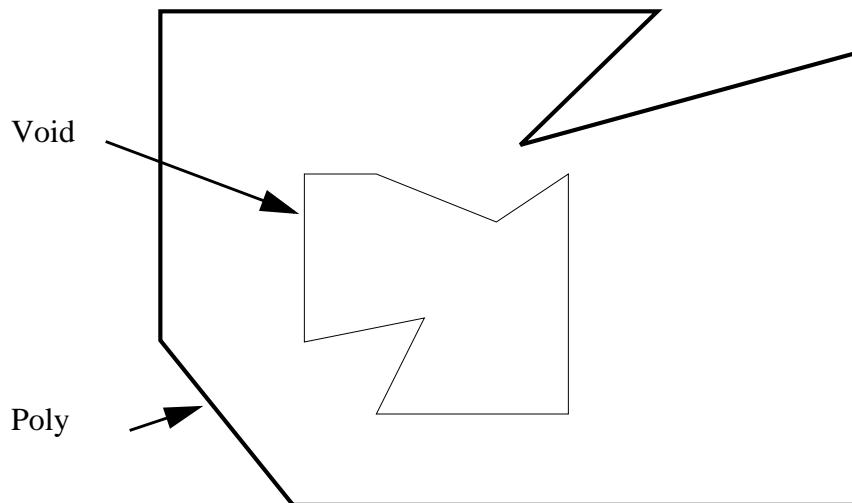
nil is returned if the operation fails. You can get a descriptive error message by calling `axlPolyErrorGet()`.

Example

```
poly_list = (axlPolyFromDB shape_dbid)
exp_poly = (axlPolyExpand poly_list 10.0 'ALL_ARC)
```

The following sequence of diagrams illustrates the behavior of each of the options.

Figure 20-2 Original Poly

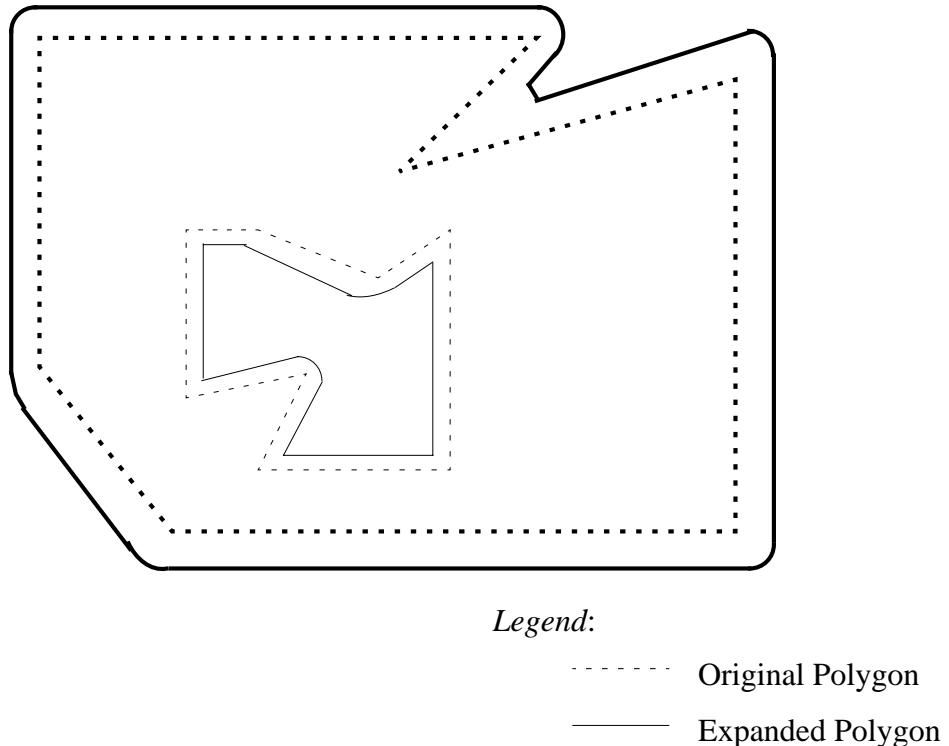


ALL_ARC

During expansion of the poly boundary, an arc is inserted for the edges in the offset shape that satisfy the following criteria:

- Edges form an *outside* (or convex) point of the poly boundary.
The reverse is true for the voids.

Figure 20-3 Expanded Using ALL_ARC

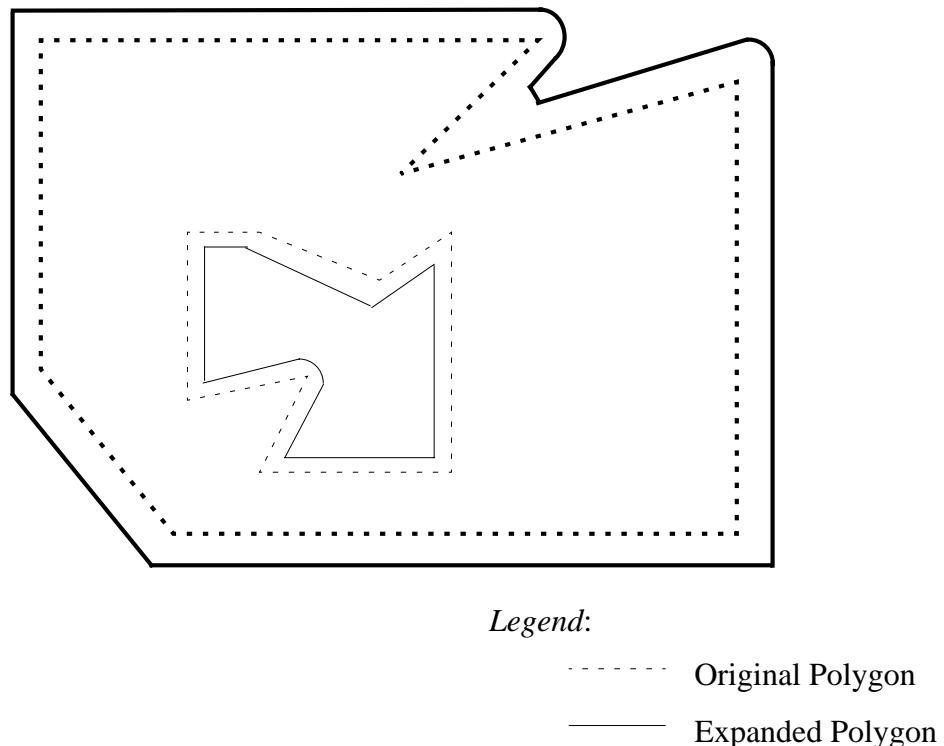


ACU_ARC

During expansion of the poly boundary, an arc is inserted for the edges in the offset shape that satisfy the following criteria:

- Edges form an *outside* (or convex) point of the poly boundary.
- Edges form an angle sharper than 90 degree.
The reverse is true for the voids.

Figure 20-4 Expanded Using ACU_ARC

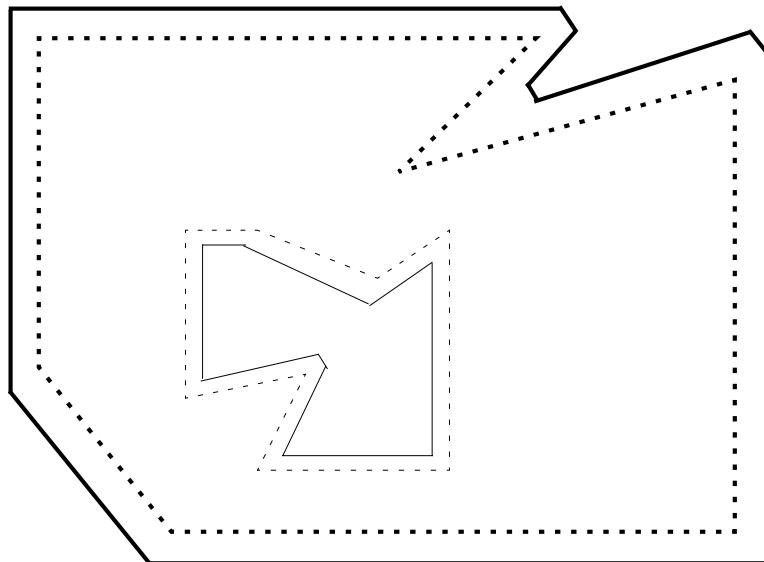


ACU_BLUNT

During expansion of the poly boundary, a blunt edge is inserted for the edges in the offset shape that satisfy the following criteria:

- Edges form an *outside* (or convex) point of the poly boundary.
- Edges form an angle sharper than 90 degree.
The reverse is true for the voids.

Figure 20-5 Expanded Using ACU_BLUNT



Legend:

- Original Polygon
- Expanded Polygon

axlIsPolyType

```
axlIsPolyType (
    g_polygon
)
⇒ t/nil
```

Description

Tests if argument *g_polygon* is a polygon user type.

Arguments

g_polygon Object to test.

Value Returned

t *g_polygon* is a polygon user type.

nil *g_polygon* is not a polygon user type.

Example

```
poly = axlPolyFromDB(cline_dbid)
axlIsPolyType(poly) returns t.
axlIsPolyType(cline_dbid) returns nil.
```

axlPolyFromHole

```
axlPolyFromHole (
  o_polygon
)
⇒ lo_polygon/nil
```

Description

Creates a new poly from the vertices of the hole, and sets the *isHole* attribute of the resulting poly to `nil`. Function returns `nil` in case of error.

Arguments

`o_polygon` `o_polygon` on which the operation is to be done. Must have *isHole* attribute set to `t` (that is, the argument must be a hole).

Value Returned

`lo_polygon` List of `o_polygons` which represent the resulting geometry after creating poly from the hole argument.

`nil` Error due to incorrect argument.

Example

```
poly = axlPolyFromDB(shape_dbid)
hole = car(poly->holes)
polyList = axlPolyFromHole(hole)
```

axlPolyErrorGet

```
axlPolyErrorGet ()  
⇒ t_error/nil
```

Description

Retrieves the error from the logop core. See the following list of error strings returned by the logical operation core:

Error type	String returned
problem with arcs	"Bad arc data in polygon operations."
bad data	"Data problem inside polygon operations."
internal error in logical op data handling	"Polygon operation failed because of internal error."
numerical problem in logical op	"Computational problem while doing polygon operations."
memory problem	"Out of memory."
no error	NIL

Arguments

None.

Value Returned

t_error Error from the logical operation core.

nil No logical operation error.

Example

```
l_poly1 = axlPolyFromDB(shape_dbid)
l_poly2 = axlPolyFromDB(cline_dbid ?endCapType 'SQUARE)
l_polyresult = axlPolyOperation(l_poly1 l_poly2 'ANDNOT)
if (null l_polyresult) axlMsgPut(list axlPolyErrorGet())
```

Use Models

Example 1

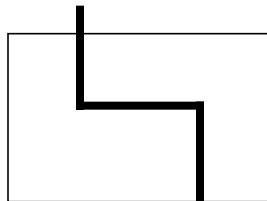
The existing Split Plane functionality can use the AXL version of the logical operation.

The objective is to split the route-keepin shape on the basis of the anti-etch information and add the resulting split-shapes to the database.

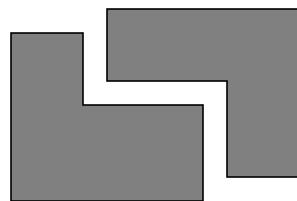
Split Plane Usage



1. startShape - route keepin rectangle



2. antiEtchGeom - anti-etch line on layer "ANTI ETCH/XYZ"



3. The above two shapes are created after doing the operation ANDNOT and then creating the shape for the resultant polys.

```
; retrieve the route keepin rectangle
startShape = (myGetRouteKeepin)

; retrieve the shapes and lines on the anti-etch subclass
antiEtchGeom = (myGetAntiEtchGeom (axlGetActiveLayer))

; create the polygon for the route-keepin rectangle
startPoly = (axlPolyFromDB startShape)
; create the polygon for all the anti-etch elements
antiEtchPoly = nil
(foreach antiElem antiEtchGeom
  antiElmPoly = (axlPolyFromDB antiElem ?endCapType 'ALL_ARC)
  antiEtchPoly = (append antiElmPoly antiEtchPoly)
)
; do the LogicalOp operation
splitPolyList = (axlPolyOperation startPoly antiEtchPoly 'ANDNOT)

;check for any error in logop
(if splitPolyList then
```

```
(; add the resultant polygons as a set of filled shapes on the
; active class/subclass with no net name.
(foreach resPoly splitPolyList
(axlDBCreateShape resPoly t)
))
else
(axlMsgPut(list axlPolyErrorGet()))
)
```

Example 2

```
; retrieve the polygon corresponding to clock gen
clockPoly = (axlPolyFromDB rect_id)
; get polygon associated with the ground shape
gndPoly = (axlPolyFromDB shp_dbid)
; get the intersection of the two polygons
shieldedPoly = (axlPolyOperation clockPoly gndPoly 'AND)

; check for any error in logop
(if shieldedPoly then
(; get the area of the intersection polygon
shieldedArea = shieldedPoly->area
; get the area associated with clock gen
clockArea = clockPoly->area

;get the ratio of the two areas
ratioArea = shieldedArea / clockArea)
else
(axlMsgPut(list axlPolyErrorGet()))
)
```

Gets the shielded area of a clock generator by a ground shape using the rule stating that the ratio of the shielded area to the actual area should be less than a particular threshold.

Allegro PCB and Package User Guide: SKILL Reference
Polygon Operation Functions

PCB Editor File Access Functions

AXL-SKILL File Access Functions

This chapter describes the AXL-SKILL functions that open and close PCB Editor files.



Use these functions, instead of SKILL's infile and outfile, to access files using PCB Editor standards, not via the SKILL path.

axlDMFileError

```
axlDMFileError(  
) -> nil/t_errorMessage
```

Description

This returns the error from the last `axlDMXXX` call. Subsequent calls reset the error message so you should retrieve the error as soon as a call fails.

Arguments

none

Value Returned

<code>nil</code>	No error message available.
<code>t_errorMessage</code>	Message indicating why last operation failed.

Example

```
q = axlDMOpenFile( "TEMP" "foo.bar" "r" )  
unless(q  
printf("ERROR is %L\n" axlDMFileError()))
```

axlDMFindFile

```
axlDMFindFile (
    t_id
    t_name
    t_mode
)
⇒ t_name/nil
```

Description

Opens a file using PCB Editor conventions. Adds an extension and optionally looks it up in an PCB Editor search path.

Note: Must have an entry in `fileops.txt` file.

Arguments

<i>t_id</i>	Id describing file attributes from <code>fileops.txt</code>
<i>t_name</i>	Name of file to find.
<i>t_mode</i>	Open mode. One of the following: r: read-only w: write wf: write line-buffered

Value Returned

<i>t_name</i>	Name of file opened.
nil	Failed to open file.

axlDMGetFile

```
axlDMGetFile(  
    t_id  
    t_name  
    t_mode  
)  
⇒ t_name/nil
```

Description

Gets the file name *t_name* using PCB Editor conventions as described in the arguments.
Returns the full path name of the file.

Arguments

<i>t_id</i>	File attribute id.
	This string must be one of the types in the PCB Editor system file <code>fileops.txt</code> . Examples are <code>ALLEGRO_LAYOUT_DB</code> for PCB Editor layouts with extension <code>brd</code> , <code>ALLEGRO_REPORT</code> for PCB Editor report files with extension <code>rpt</code> .
<i>t_name</i>	String giving name of the file to open.
<i>t_mode</i>	Open mode. One of the following: <code>r</code> : read-only <code>w</code> : write <code>wf</code> : write line-buffered

Value Returned

<i>t_name</i>	Name of the file. In the case of <i>t_mode</i> = "r", the file must exist for successful completion.
<i>nil</i>	File not found. Displays a confirmmer giving the name of the file it could not find.

Example

```
myfile = axlDMGetFile( "ALLEGRO_TEXT" "clip" "r")
⇒ "/usr/home/fred/myproj/clip.txt"
```

Finds the file `clip.txt`, available for reading.

axlDMOpenFile

```
axlDMOpenFile(  
    t_id  
    t_name  
    t_mode  
)  
⇒ p_port/nil
```

Description

Opens a file in convention Allegro manner; adds an extension and optionally looks it up in a Allegro search path. Must have an entry in fileops.txt file.

Allegro currently does not support directory or file names containing spaces.

Use this in place of Skill's infile/outfile. The Skill interfaces resolve the file location using SKILLPATH which may mean that files may not open in the local directory if the SKILLPATH does not have "." as its first component. axlDMOpenFile uses the Allegro convention to open file.

Note if you use axlDMOpenFile to open a file, use axlDMClose to close it. All other Skill file APIs work on the port returned by this interface.

If you want to use Allegro's standard file extension support (we will append the extension if not present), then see <cdsroot>/share pcb/text/fileops.txt for a list of t_ids.

Otherwise, if you always provide an extension, use the TEMP id.

Arguments

<i>t_id</i>	File attribute id. This string must be one of the types in the PCB Editor system file fileops.txt. Examples are ALLEGRO_LAYOUT_DB for PCB Editor layouts with extension brd, ALLEGRO_REPORT for PCB Editor report files with extension rpt.
<i>t_name</i>	String giving the name of the file to open.
<i>t_mode</i>	Open mode. One of the following: r: read-only

w: write

a: open for writing, create if doesn't exist, go to end of file for appending if exists

In addition, the following modifiers are supported

f: Flush file after each write. This can be slow on Windows if writing across the network. This is typically used if a process will take a long time and the user would like to look at the file to see the progress. Example "wf"

b: Open in binary mode. This only has effect on Windows. If file is Ascii has the effect for reading of not eliminating the carriage-return (\r) that are in DOS Ascii files. For writing doesn't add the carriage-return when it sees a linefeed (writes it like a UNIX Ascii file). Example "rb"

s: Allow spaces in file or directory name. Currently Allegro doesn't support this behavior. Setting this option unsupported. Example "rbs"

Value Returned

p_port Port of the opened file. In the case of *t_mode* = "r", the file must exist for successful completion.

nil File not found. Displays a confirmor giving the name of the file it could not find.

See also

[axlDMFileError](#)

Example

- opens a file; clip.txt for writing

```
aPort = axlDMOpenFile("ALLEGRO_TEXT" "clip" "w")
```

- opens a file; b.bar

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor File Access Functions

```
aPort = axlDMOpenFile( "TEMP" "foo.bar" "r" )
```

axlDMDOpenLog

```
axlDMDOpenLog(  
    t_program  
)  
⇒ p_port/nil
```

Description

Opens a file for writing log messages. Uses the name of your program or application without an extension. Opens a file with that name and the extension .log. Returns the port of the file if it succeeds.

Arguments

t_program Your program name - no extension.

Value Returned

p_port Port of the opened file. In the case of *t_mode* = "r", the file must exist for successful completion.

nil File not found. Displays a confirmator giving the name of the file it could not find.

Example

```
logport = axlDMDOpenLog("clipboard") )
```

Opens the file clipboard.log for writing.

axlDMClose

```
axlDMClose(  
    p_port  
)  
⇒ t/nil
```

Description

Closes an opened PCB Editor file. While you may close a file via the core SKILL function, `close`. It is suggested that any file opened via an `axlDM` function be closed via this function. If your program adheres to this standard then it will be compatible with future Allegro Data Management enhancements.

Use this in place of Skill's `infile/outfile` if you have used `axlDMOpenFile` or `axlDMOpenLog`.

Arguments

p_port Id of the open port to be closed.

Value Returned

t Closed the file.

nil File not found.

Example

```
mylog = axlDMOpenLog("myapplic")  
      ⇒ port: "/usr/home/fred/myproj/myapplic.log"  
axlDMClose(mylog)  
      ⇒ t
```

Opens and closes the file `myapplic.log`.

axlDMBrowsePath

```
axlDMBrowsePath(  
    t_adsFileType  
    [t_title]  
    [t_helpTag]  
)  
⇒ t_fileName/nil
```

Description

Invokes a standard PCB Editor file browser supporting paths, for example, SCRIPTPATH. To use, pass one of the file types supported by `fileops.txt`. Browses file types that include the `fileops PATH` attribute. `axlDMFileBrowse` should be used to browse other file types. This works on non-PATH file types since this browses in the current working directory. The user is not able to change the directory with this browser.

Arguments

<code>t_adsFileType</code>	First entry in <code>fileops.txt</code> .
<code>t_title</code>	Title for the dialog
<code>t_helpTag</code>	Tag for the help file (used only by Cadence)

Value Returned

<code>t_filename</code>	Full path to the filename.
<code>nil</code>	Error due to incorrect arguments.

Example

```
ret = axlDMBrowsePath( "ALLEGRO_SCRIPT" )  
ret = axlDMBrowsePath( "ALLEGRO_CLIPBOARD" "Select Clipboard" )
```

axlDMDirectoryBrowse

```
axlDMDirectoryBrowse(  
    t_startingDirectory  
    g_writeFlag  
    [?helpTag t_helpTag]  
    [?title t_title]  
)  
⇒ t_dirName/nil
```

Description

Opens a directory browser. Unlike file browsers, this only allows a user to select a directory. This function call blocks until the user selects or cancels.

Arguments

<i>t_startingDirectory</i>	Name of the starting directory.
<i>g_writeFlag</i>	A boolean - if the file is to be opened for write (<i>t</i>), or for read (<i>nil</i>).
<i>t_helpTag</i>	Defines the help message to display if the <i>Help</i> button is selected in the browser. Default help is provided if this option is not set.
<i>g_title</i>	Override default title bar of the browser. Normally, this is the name of the command that invoked the browser.

Value Returned

<i>t_dirName</i>	Name of directory selected.
<i>nil</i>	No directory selected.

Example

```
axlDMDirectoryBrowse( ". " t ?title "Pick a directory")
```

Browses the current directory.

axlDMFileBrowse

```
axlDMFileBrowse(
    t_fileType
    g_writeFlag
    [?defaultName t_defaultName]
    [?helpTag t_helpTag]
    [?directorySet g_directorySet]
    [?noDirectoryButton g_noDirectoryButton]
    [?mainFile g_mainFile]
    [?noSticky g_noSticky]
    [?title t_title]
    [?optFilters t_filters]
)
⇒ t_fileName/nil
```

Description

Opens a standard file browser. Unlike the other `axlDM` functions, this always presents the user with a file browser. This function call blocks until the user selects a file or cancels.

Note: The name of the file is selected and returned to the caller. Does not open the selected file.

The final filter is 'All files (*.*)'.

Arguments

<code>t_id</code>	Id describing the file attributes from <code>fileops.txt</code> , or list of ids for different types, or <code>nil</code> if you use <code>optFilters</code> to describe files.
<code>g_writeFlag</code>	If the file is to be opened for write (<code>t</code>), for read (<code>nil</code>).
<code>t_defaultName</code>	Name of file to select by default.
<code>t_helpTag</code>	Tag that defines the help message to display if the Help button is selected in the browser. Default help provided if option not set.
<code>g_directorySet</code>	Sets the directory change button which, by default, is not set.
<code>g_noDirectoryButton</code>	Hiding of the directory change button in the browser. By default, the button is present.

Allegro PCB and Package User Guide: SKILL Reference

PCB Editor File Access Functions

<i>g_noSticky</i>	File browser normally remembers the directory from the previous invocation. This helps the user who browses in the same location that is different from the current working directory. If t, then it starts the browser in the current working directory. Normally, you should set this option if <i>g_directorySet</i> is t.
<i>g_mainFile</i>	Matches options PCB Editor uses to open files from the File menu. This is <i>g_noSticky=t & g_directorySet=t</i> . For non-main files, use no options.
<i>g_title</i>	Overrides default title bar of the browser. Normally this is the name of the command that invoked the browser.
<i>g_filters</i>	Filters added to default <i>t_id</i> filter. The format is: <i><msg> <filter> <msg> <filter>...</i>

Value Returned

<i>t_fileName</i>	Name of the file selected.
<i>nil</i>	No file selected.

Example 1

```
axlDMFileBrowse( "ALLEGRO_TEXT" nil )
```

Browses PCB Editor text files.

Example 2

```
axlDMFileBrowse( "ALLEGRO_TEXT" nil ?optFilters "All log files|*.log|")
```

Browses PCB Editor text files and allows secondary filter of *.log.

Example 3

```
axlDMFileBrowse( nil nil ?optFilters "Skill files(*.il)|*.il|")
```

Browses SKILL files.

axlDMFileParts

```
axlDMFileParts(  
    t_filespec  
)  
⇒ (directory file fileWext ext)
```

Description

Breaks a filename into it's component parts.

Arguments

t_filespec Filename or full path spec.

Value Returned

list (*directory file fileWext ext*)

Example

```
fileparts = axlDMFileParts( "/usr1/xxx/stuff.txt" )  
    --> ( "/usr1/xxx/" "stuff" "stuff.txt" "txt" )  
  
fileparts = axlDMFileParts( "stuff.txt" )  
    --> ( "/usr1/xxx/" "stuff" "stuff.txt" "txt" )  
  
    **where /usr1/xxx is the cwd
```

axlOSFileCopy

```
axlOSFileCopy(  
    t_src  
    t_dest  
    g_append  
)  
⇒ t/nil
```

Description

Copies a given source file to a given destination with optional append.

Arguments

<i>t_src</i>	Full path of the source file.
<i>t_dest</i>	Full path of the destination file.
<i>g_append</i>	Flag for the append function (<i>t/nil</i>)

Value Returned

<i>t</i>	Copied file.
<i>nil</i>	Failed to copy file due to incorrect arguments.

Example

```
unless(axlOSFileCopy( "~/myfile" "~/newfile" nil)  
    axlUIConfirm("file copy FAILED") )
```

axlOSFileMove

```
axlOSFileMove(  
    t_src  
    t_dest  
)  
⇒ t/nil
```

Description

Moves the given source file to the given destination.

Arguments

t_src Full path of the source file.

t_dest Full path of the destination file.

Value Returned

t Moved file.

nil Failed to move file.

Example

```
unless (axlOSFileMove( "/mydir/myfile"  "/newdir/newfile" )  
    axlUIConfirm("file move FAILED" ) )
```

axLOSSlash

```
axLOSSlash(  
    t_directory  
)  
⇒ t_directory/nil
```

Description

Changes DOS style backslashes to UNIX style slashes which are more amenable to SKILL. On UNIX, returns the incoming string.

Arguments

t_directory Given directory path.

Value Returned

t_directory Directory path using UNIX style slashes (/).

nil Failed due to incorrect argument.

Example

```
p = axLOSSlash( "\tmp\mydir" )  
-> "/tmp/mydir"
```

axlRecursiveDelete

```
axlRecursiveDelete(  
    t_directory  
)  
⇒ t/nil
```

Description

Recursively removes directories and subdirectories in the argument list. Directory is emptied of files and removed. If the removal of a non-empty, write-protected directory is attempted, the utility fails. If it encounters protected files or sub-directories, it does not remove them or the parent directories, but removes all other objects.



This can be dangerous since it can severely damage your system or data if not used with care. For example, axlRecursiveDelete("/") could delete your OS and all of your data.

Arguments

t_directory The given directory or filename.

Value Returned

t Directory is successfully removed.

nil Failed for one of the following reasons:
- doesn't exist
- read protected
- sub-file or directory does not allow remove (*partial success*)
- sub-file or directory is in use (NT only) (*partial success*)
A partial success means that some of the files and directories were deleted.

Example

```
parent = "./tmp"
child = (strcat parent "/child")
(createDir parent)
(createDir child)
(axlOSFileCopy"~/.cshrc" (strcat parent"/csh") nil)
(axlOSFileCopy"~/.cshrc" (strcat child"/csh") nil)
(axlRecursiveDelete parent)
```

axlTempDirectory

```
axlTempDirectory(  
    )  
⇒ t_directoryName/nil
```

Description

Returns the temporary directory for the current platform.

Arguments

none

Value Returned

<i>t_directory</i>	Temporary directory for the current platform.
<i>nil</i>	Failed to identify temporary directory for the current platform.

axlTempFile

```
axlTempFile(  
    [g_local]  
)  
⇒ t_tempFileName/nil
```

Description

Returns a unique temp file name. The temp file should be removed, even if not used, by axlTempFileRemove.

By default, the files are written to /tmp, but you can modify this with the environment variable TEMPDIR.

Arguments

<i>g_local</i>	Flag, which if t, creates a temp file in the current directory. Most applications should use the default /tmp directory. The local directory should only be used if the file will be more than 2 megabytes.
----------------	---

Value Returned

<i>t_tempFileName</i>	Name of the unique temp file.
-----------------------	-------------------------------

nil	Failed to create temp file.
-----	-----------------------------

axlTempFileRemove

```
axlTempFileRemove(  
    t_filename  
)  
⇒ t
```

Description

Deletes the temporary file and removes the temporary name from the pool. It is important to call this function once you are finished with a temporary filename.

This can also be used to delete files whose names are not obtained from axlTempFile.

Arguments

t_filename Name of the file to delete.

Value Returned

t Deleted temporary file specified.

nil Failed to delete temporary file specified due to incorrect argument.

Allegro PCB and Package User Guide: SKILL Reference
PCB Editor File Access Functions

Reports and Extract Functions

AXL-SKILL Data Extract Functions

The chapter describes the AXL-SKILL functions that extract data from an Allegro layout and write it to an ASCII file for external processing.

axlExtractToFile

```
axlExtractToFile(  
    t_viewFile  
    lt_resultFiles  
    [lt_options]  
)  
⇒ t/nil
```

Description

Extracts data from the current design into an ASCII file. Performs the same process as the Allegro batch command `a_extract`.

Arguments

<i>t_viewFile</i>	Name of the extract command file (view file).
<i>lt_resultFiles</i>	String or list of strings giving the names of the files to which to write the extracted ASCII data. If you designate only one output file, this is a string.
<i>lt_options</i>	List of keyword strings for various options:
"crosshatch"	Generate crosshatch lines when extracting crosshatched shapes.
"ok"	Unknown field names are OK. Useful when the extract command file has property names not defined for the design being extracted.
"short"	Extract data in the short format
"quiet"	Do not print progress messages on stdout.
"mcm"	Output MCM terminology.

Value Returned

<i>t</i>	Extract complete.
<i>nil</i>	Failed to complete the extract. See the file <code>extract.log</code> for explanatory error messages.

Example

```
a xlExtractToFile( "cmp_rep_view" "my_comp_data" )  
⇒ t
```

Extracts the component data from the current layout. Writes the data to the file `my_comp_data.txt`.

axlExtractMap

```
axlExtractMap(  
    t_viewFile  
    [s_applyFunc]  
    [g(userData)]  
)  
⇒ t/l_dbid/nil
```

Description

Takes a set of Allegro database objects you select using the Allegro extract command file and applies to each object a SKILL function you have chosen. The extract command file (the *view*) selects the objects and also sets any filters. Ignores the output data fields listed in the extract command file, since it does not create an output file.

Applies to each object that passes the selection and filter criteria in *s_applyFunc*, the SKILL function you supplied. SKILL calls *s_applyFunc* with two arguments—the *dbid* of the object and the variable *g.userData* that you supply as an argument. *g.userData* may be *nil*. If not, it normally is a *defstruct* or disembodied property list so that the called routine can update it destructively.

If *s_applyFunc* returns *nil*, then *axlExtractMap* stops execution, writes the message, **User CANCEL received** to the extract log file, and returns *nil*. If the callback function returns non *nil* then execution continues.

If *s_applyFunc* is *nil*, then *axlExtractMap* simply returns *l_dbid*, a list of *dbids* of the objects that pass the filter criteria. Use *l_dbid* to perform a *foreach* to operate on each object. (See examples below.)

Behaves slightly differently from a standard extract to a file. Provides *s_applyFunc* with the *dbid* of the parent (rotated) rectangle or shape, and not individual line/arc segments. Returns the attached text in the **FULL GEOMETRY** view.

Allegro “pseudoviews” (DRC, ECL, ECS, ECL_NETWORK, PAD_DEF, and so on) may not be used with *axlExtractMap*. The callback function is never called.

Note: *axlExtractMap* extracts only objects AXL-SKILL can model. For example, it does not extract function instances and unplaced components.

Arguments

<i>t_viewFile</i>	Name of the extract command file (view file).
<i>s_applyFunc</i>	SKILL function to call for each selected object.
<i>g(userData)</i>	Data to pass to <i>s_applyFunc</i> . May be nil. Ignores <i>g(userData)</i> if <i>s_applyFunc</i> is nil.

Value Returned

<i>t</i>	<i>s_applyFunc</i> is non nil. Applied SKILL functions to specified objects.
<i>l_dbid</i>	List of <i>dbids</i> of the specified objects.
<i>nil</i>	Failed to apply SKILL functions to specified objects. See the file <i>extract.log</i> for explanatory error messages.

Example

```
; user callback provided - function declared
; with the (lambda) function
; returns a list of the names of all nets
(defun get_netnames ()
  ; byReference is disembodied prop list
  ; with property 'nets to be used to store
  ; the net names
  (let ((byReference (list nil 'nets nil)))
    (axlExtractMap "net_baseview"
      ; function created right here
      (lambda (dbid netRef)
        ; add the name of this net to
        ; the list of names
        (putprop netRef
          (cons (get dbid 'name)
            (get netRef 'nets)) 'nets)
        t) ; success return
      byReference)
    (get byReference 'nets))) ; return list of names
```

axlReportList

```
axlReportRegister(  
    ) -> ll_reportList/nil
```

Description

Lists all the SKILL reports registered to the PCB Editor report interface. Even if you do not register any reports, PCB Editor registers default reports written in SKILL.

Arguments

None.

Value Returned

ll_reportList List of lists of reports register.

Each sub-list has the following format:

(g_reportCallback t_description t_title)

nil No reports registered.

See also `axlReportRegister`.

axlReportRegister

```
axlReportRegister(  
    g_reportCallback  
    t_description  
    t_title  
) ==> t
```

Description

Allows registration of user reports using the PCB Editor report dialog box. You provide a report name, title, and a report-generating callback function. Callback function format is: `g_reportCallback(t_outfile)`.

If you generate the report, returns a `t` from the function, or `nil`, if you do not generate a report. If you provide a `nil` `t_description` then the current report is unregistered. You can disregard the `t_title` in the unregister mode. You can register only one report per callback function (`g_reportCallback`). To delete an existing report, pass the callback function assigned to the report, a `nil` for the `t_description`, and an empty string for the `_title`.

Note: Only applicable if you enable the *HTML-style reports*.

You can generate the report file in two formats: text or HTML.

Text:

- File is text based.
- Conversions include inserting links when following keywords are encountered:
`(http://)` - add an HTML link
`(num <,> num)` - XY coordinate; add a cross-probe link to zoom center on that coordinate

HTML:

- File starts with an `<HTML>`
- To enable xy zoom center, decorate all xy coordinates as follows:
`<a href="xprobe:xy:<x>,<y>">(x y)`
Example: xy coordinate is (310 140)
`(310.0 140.0)`

Suggestions and restrictions:

- Your report description string should start with your company name or some other standard to keep all your reports together in the report dialog. Keep the description short to fit within the space provided in the dialog box.
- The report dialog is blocking. Do not use the following:

APIs:

- any user pick or selection API, `axlSelect`, `axlEnter`.
- `axlShell`
- save or open a drawing.

- You can invoke your own form within the report callback but make sure it is blocking (use `axlUIWBlock(<formhandle>)`)
- Return `t` if you want the report to display. A `nil` will not display the report
- If building a context where your report function is stored, you cannot make the `axlReportRegister` inside the context. The `axlReportRegister` must be placed in `allegro.ilinit` or the `.ini` file (if building autoload contexts). This is the same rule that applies to `axlCmdRegister`.

Arguments

`g_reportCallback` Symbol or string of a SKILL function.

`t_name` Name of report (shown in reports dialog box); if `nil` will delete the report associated with `g_reportCallback`.

`t_title` Name in title bar when displaying report.

Value Returned

`t` Arguments correct.

`nil` Illegal argument.

Examples

The following registers the report My Hello. The optional keyword in MyReport lets a direct call to MyReport generate a report to a fixed name file, `helloWorld.rpt`. otherwise it takes the name from the report dialog box.

```
axlRegisterReport('MyReport, "XYZ Inc. Hello" "Hello World")  
  
; optional keyword allows you to call MyReport with a nil where  
; the report file generated will be helloWorld.rpt. Otherwise if  
; called from report dialog, it will be passed a temporary filename  
procedure( MyReport(@optional (rptName "helloWorld"))  
  
let( (fp)  
fp = axlDMOpenFile("ALLEGRO_REPORT" reportFile "w")  
axlLogHeader(fp "This is my report")  
fprintf(fp, "\nHello World\n")  
close(fp)  
t  
) )
```

Unregister above report:

```
axlRegisterReport('MyReport, nil nil)
```

See also `axlRegisterList`, `axlLogHeader`.

Utility Functions

Overview

The chapter describes the AXL-SKILL utility functions. These include functions to derive arc center angle and radius, and to convert numeric values to different units.

axlCmdList

```
axlCmdList()  
)  
⇒ ll_strings/nil
```

Description

Lists commands registered by `axlCmdRegister`. The list consists of paired strings.

```
((allegro command <skill function>)...)
```

Arguments

None.

Value Returned

<i>ll_strings</i>	List of registered PCB Editor commands paired with the related SKILL functions.
<i>nil</i>	No commands registered.

Examples

```
axlCmdRegister( "interboxcmd" 'axlEnterBox)  
axlCmdList()  
    will return the following:  
(  
    ( "interboxcmd" "axlEnterBox" )  
)
```

axlDebug

```
axlDebug(  
    t/nil  
) t/nil
```

Description

This enables/disables AXL debug mode. See axlIsDebug for description of what this entails.

Arguments

t	To enable AXL Skill debug mode.
nil	To disable.

Values Returned

Returns last setting

See also

[axlIsDebug](#)

axlEmail

```
axlEmail(  
    t_to  
    t_cc/nil  
    t_subject  
    t_body  
)  
==> t/nil
```

Description

Sends an e-mail. If multiple *To* or *Cc* addresses are required; separate their names with a semicolon (;). On Windows, a warning confirmmer may generate. To disable the warning, add the following to the registry:

```
HKEY_CURRENT_USER/Identities/[ident code]/Software/  
Microsoft/Outlook Express/5.0/Mail/Warn on Mapi Send
```

Set the data value to 0.



On UNIX, it is not possible for the interface to return an e-mail delivery failure.

Arguments

<i>t_to</i>	E-mail address.
<i>t_cc</i>	Any carbon copy persons to send; <i>nil</i> if none.
<i>t_subject</i>	What to put on the subject line.
<i>t_body</i>	Body of message.

Value Returned

<i>t</i>	If successful.
<i>nil</i>	Failure.

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

Example

```
axlEmail( "aperson" nil "A test message" "anybody home?")
```

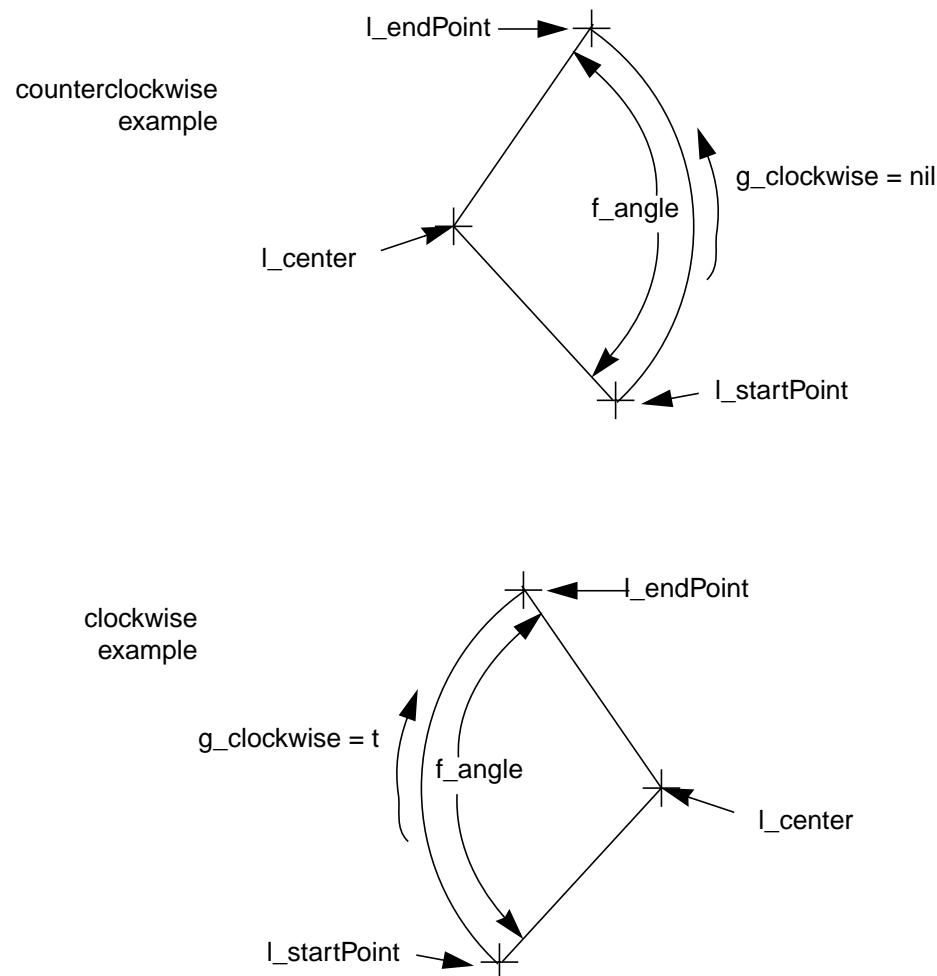
axlGeoArcCenterAngle

```
axlGeoArcCenterAngle(  
    l_startPoint  
    l_endPoint  
    f_angle  
    [g_clockwise]  
)  
⇒ l_center/nil
```

Description

Calculates the center of an arc given the angle between its endpoints. Uses the arguments depending on *g_clockwise* as shown in [Figure 23-1](#) on page 825.

Figure 23-1 Center of Arc Calculation



Arguments

<i>l_startPoint</i>	Start point of the arc
<i>l_endPoint</i>	End point of the arc
<i>f_angle</i>	Included angle of the arc
<i>g_clockwise</i>	Rotational sense of the arc: <code>t</code> is clockwise. <code>nil</code> is counterclockwise (the default).

Value Returned

l_center Center of the arc as a list: (*X Y*).

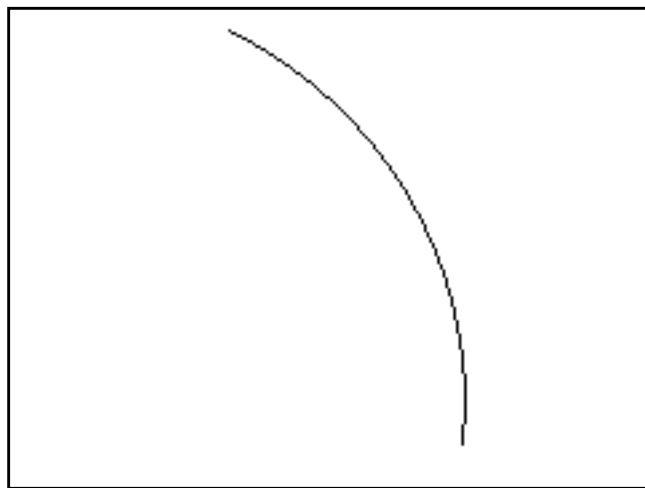
nil Cannot calculate the center from the given arguments.

Example

```
print axlGeoArcCenterAngle( 7500:5600 8000:4700 68.5 t)
mypath = axlPathStart(list( 7500:5600))
    axlPathArcAngle(mypath, 0., 8000:4700, t, 68.5)
    axlDBCreatePath( mypath, "etch/bottom")
⇒ (7089.086 4782.826)
```

Prints the center for the clockwise arc going through the points (7500:5600) and (8000:4700) using `axlGetArcCenterAngle`, then adds the arc through those points using `axlPathArcAngle`, and compares their centers.

The arc is shown in the following figure.



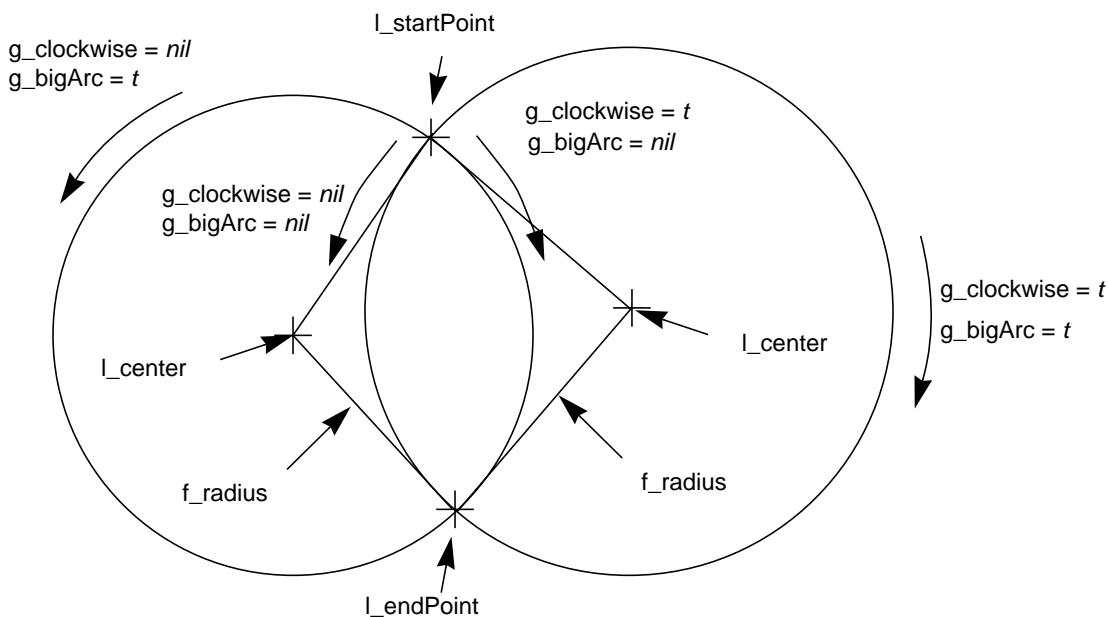
Do `Show Element` on the arc to show its center coordinate. The arc lists: “center-xy (7089, 4783)” which agrees with the (7089.086 4782.826) printed by `axlGeoArcCenterRadius`.

axlGeoArcCenterRadius

```
axlGeoArcCenterRadius(  
    l_startPoint  
    l_endPoint  
    f_radius  
    [g_clockwise]  
    [g_bigArc]  
)  
⇒ l_center/nil
```

Description

Calculates center of an arc given its radius. Calculates *l_center* either for one arc or another depending on the arguments, as shown.



Arguments

<i>l_startPoint</i>	Start point of the arc
<i>l_endPoint</i>	End point of the arc
<i>f_radius</i>	Radius of the arc

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

g_clockwise Rotational sense of the arc: *t* is clockwise. *nil* (default) is counterclockwise.

g_bigArc Flag telling whether the arc extends as the larger or the smaller of the two arcs possible between the start and endpoints.

Value Returned

l_center Center of the arc as a list: (*X Y*).

nil Cannot calculate the center from the given arguments.

Example

```
print axlGeoArcCenterRadius( 7500:5600 8000:4700 1000)
⇒ (8499.434 5566.352)

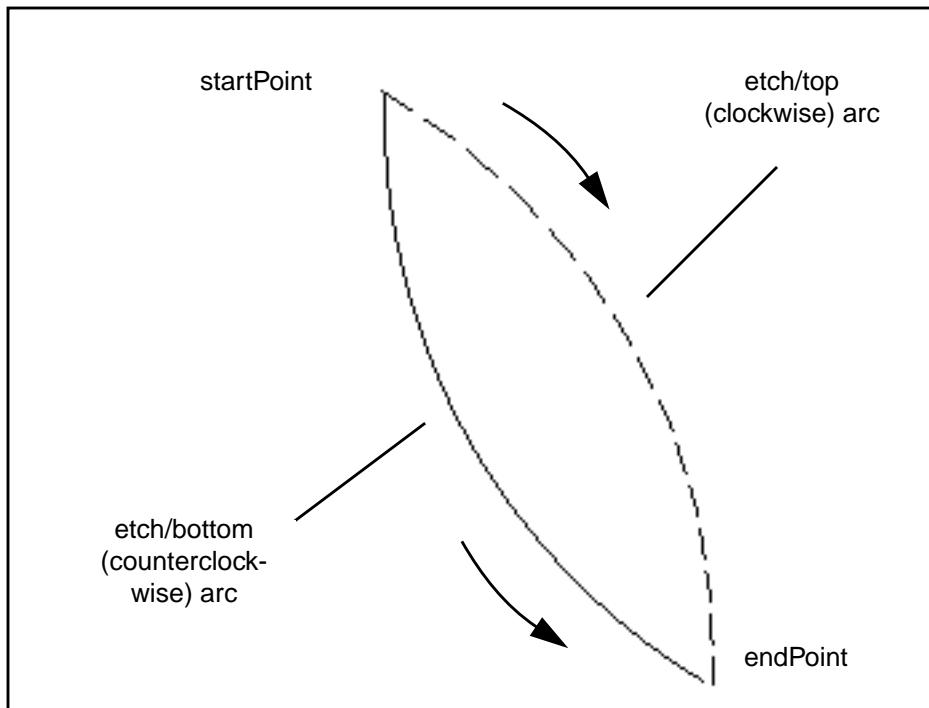
mypath = axlPathStart(list( 7500:5600))
axlPathArcRadius(mypath, 0., 8000:4700, t, nil, 1000)
axlDBCreatePath( mypath, "etch/bottom")

print axlGeoArcCenterRadius( 7500:5600 8000:4700 1000 t)
⇒ (7000.566 4733.648)

mypath = axlPathStart(list( 7500:5600))
axlPathArcRadius(mypath, 0., 8000:4700, nil, nil, 1000)
axlDBCreatePath( mypath, "etch/top")
```

Prints the two possible centers for the arcs going through the points (7500:5600) and (8000:4700), then adds arcs through those points using `axlPathArcRadius`, and compares the centers.

The arcs are shown in the following figure.



Do *Show Element* on each arc to show its center coordinate. The solid arc on the left lists: "center-xy (8499,5566)" which agrees with the (8499.434 5566.352) printed by `axlGeoArcCenterRadius`. The dotted arc on the right lists: "center-xy (7001, 4734)", which agrees within rounding with (7000.566 4733.648) printed for the other arc.

Arguments 3

```
axlMKSCConvert(  
    nil  
    [t_outUnits]  
)  
⇒ t/nil
```

Pre-registers *t_outUnits*, the input units string, so that subsequent calls to axlMKSCConvert need not specify units.

<i>t_outUnits</i>	String giving the units to convert to for subsequent calls to axlMKSCConvert. If there is no active drawing, function fails with this combination of arguments.
-------------------	---

Value Returned 3

t	Conversion string acceptable.
nil	Conversion string not acceptable.

Example 3

See Example 4 below.

Arguments 4

```
axlMKSCConvert(  
    n_input  
)  
⇒ f_output/nil
```

Use this combination of arguments only after a call to axlMKSCConvert as in Arguments 3. Converts the number *n_input* specifying a value using the *t_outUnits* supplied by a previous call to axlMKSCConvert, and returns as *f_output*.

<i>n_input</i>	Number giving the input value to convert
----------------	--

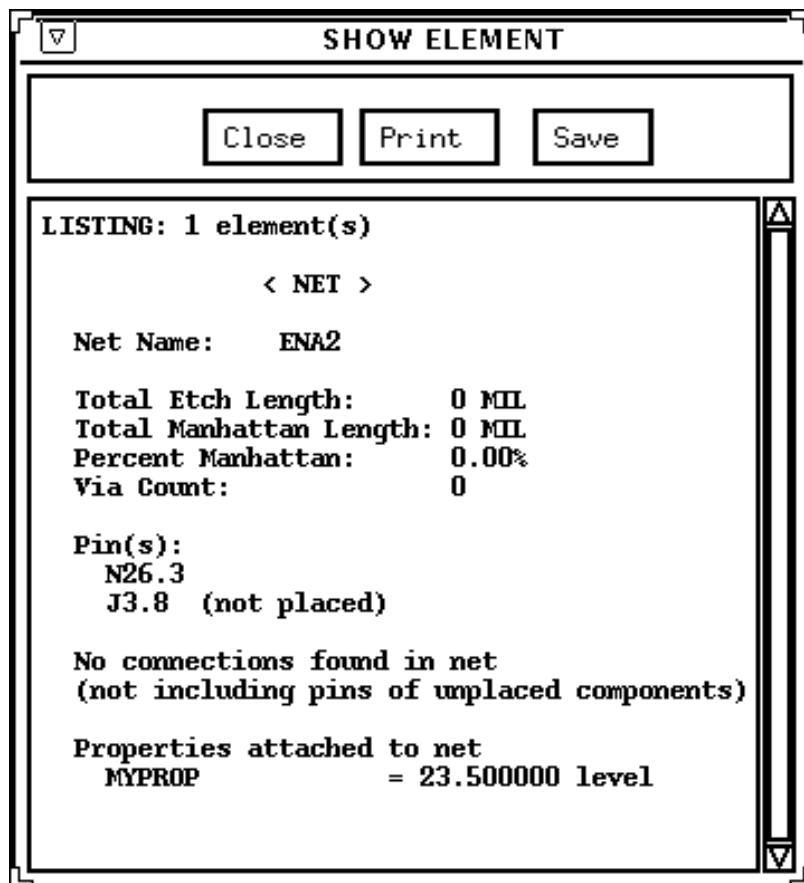
Value Returned 4

<i>f_output</i>	Converted value of <i>n_input</i> .
-----------------	-------------------------------------

Example 4

```
(axlMKSCConvert .5) -> nil ;error,if no preregistered units  
(axlMKSCConvert nil "MILS) -> t  
(axlMKSCConvert .5) -> 0.0005 ; remembers MILS
```

The following **Show Element** form shows the net with MYPROP attached:



axlHttp

```
axlHttp(  
    t_url  
)  
⇒ t
```

Description

Displays a URL from PCB Editor in an external web browser. First attempts to use the last active window of a running web browser, raising the browser window to the top if required. If no browser is running, tries to start one.

On UNIX, supports only the browser, Netscape. If Netscape is not running, tries to start it. May not detect failure if the web page fails to load.

Note: Netscape must be in your search path.

You can override the program name using the environment variable:

```
set http_netscape = <program name>
```

For example:

```
set http_netscape = netscape405
```

Note: On UNIX, this function has been tested with Netscape only and may not function properly with another web browser.

On Windows, this function uses the default web browser listed in the Windows registry. If no browser is registered, this function fails. Both Netscape and Windows Explorer work with this function. You can open any file type with a Windows registry entry.

Arguments

t_url URL to display.

Value Returned

- | | |
|-----|---|
| t | URL displayed in web browser. |
| nil | Failure due to web browser not being found (on UNIX), not being registered (on Windows), or being unable to load the URL. |

Note: The function may not detect when a URL has not loaded on UNIX.

Examples

```
axlHttp( "http://www.cadence.com" )
```

axlIsDebug

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

Description

This checks if AXL debug mode is enabled. This is associated with the `axldebug` Allegro environment variable.

When this mode is set, AXL may print additional AXL API argument warnings when arguments to AXL functions are incorrect. Many warnings do not obey this variable because they are considered serious or edge conditions. Warnings supported are less serious and are known from user feedback to be troublesome to program around.

Typically, when an AXL API function encounters an argument error, it will print a warning and return `nil`.

When unset, the code runs in development mode and eliminates many of these warnings. You should have this mode enabled when developing Skill code.

This functionality was introduced in 15.2. Currently, few AXL functions take advantage of this option.

Arguments

None

Value Returned

`t` if mode is enabled, `nil` otherwise.

See also

`axlDebug`

Example

```
when( axlIsDebug() printf( "Error\n" ) )
```

axlLogHeader

```
axlLogHeader(  
    p_port  
    t_titleString  
)  
⇒ t/nil
```

Description

Writes the standard PCB Editor log file header to the passed open file. The header record includes:

- Title String
- Drawing Name
- Software Release
- Date and Time

Arguments

p_port SKILL port for the open file.

t_titleString Title string in the log file header.

Value Returned

t Wrote log file header to open file.

nil Failed to write log file header to open file due to incorrect arguments.

axIMKS2UU

```
axIMKS2UU(  
    t_mksString  
)  
⇒ f_value/nil
```

Description

Converts between an MKS string to the current database user units. String can be any length name plus many common abbreviations (see `units.dat` file in the PCB Editor share/text hierarchy for supported names.)

Conversion can fail for the following reasons:

- Input string not a legal MKS format.
- Conversion will overflow the maximum allowed database size.

Notes:

- Conversion between metric and english units may result in rounding.
- Return number is rounded to the database precision.

Arguments

t_mksString Input unit's string with MKS units.

Value Returned

f_value Floating point number.

nil Conversion failed. See previous description for possible reasons.

Example 1

`axlMKS2UU("100.1") -> 100.0`

Default conversion with database in mils.

Example 2

`axlMKS2UU("100 mils") -> 100.0`

Database is in mils.

Example 3

`axlMKS2UU("100 inches") -> 100000.0`

Database is in mils.

Example 4

`axlMKS2UU("100 METER") -> 3937008.0`

Database is in mils.

axlMKSAlias

```
axlMKSalias(  
    t_MKSalias  
)  
⇒ t_alias/nil
```

Description

Searches the MKS unit database for the current definition associated with *unitName*. Unlike axlMKSConvert, this function does not convert.

You load the MKS database from the following file:

```
<cds_root>/share pcb/text/units.dat
```

Argument

t_mksAlias Name of the alias string.

Value Returned

t_def Definition name as a string.

nil Definition name could not be found.

Examples

```
axlMKSalias("VOLTAGE") -> "V" (Intended usage)  
axlMKSalias("M") -> "METER"  
axlMKSalias("KG") -> NIL (The function supports only the use of basic units.)
```

axlMKSCConvert

```
axlMKSCConvert(  
    n_input  
    t_inUnits  
    [t_outUnits]  
)  
=> f_output/nil
```

Description

Converts any allowable units to any other allowable units. Operates in several ways, depending on the arguments.

Note: The synopsis above shows only one of several possible argument combinations. The descriptions below detail each combination.

In all cases where this function actually does a conversion, returns the converted value as *f_output*. If it cannot convert the number for any reason, it returns *nil*. The one set of arguments to `axlMKSCConvert` that allow *nil* is the input units pre-register call, described in [Arguments 3](#).

Arguments 1

```
axlMKSCConvert(  
    n_input  
    t_inUnits  
    [t_outUnits]  
)  
-> f_output/nil
```

Converts the number *n_input* from *t_inUnits* to *t_outUnits* and returns it as *f_output*.

n_input Number to convert

t_inUnits String giving the units of the input number

t_outUnits String giving the new units for *f_output*. If *t_outUnits* is *nil*, converts the number to the current units of the layout.

Value Returned 1

f_output Converted value of *n_input*.

nil Failed to convert value due to incorrect arguments.

Example 1

```
(axlMKSConvert .5 "MILS" "INCHES") -> 0.0005
```

Arguments 2

```
axlMKSConvert(  
    t_input  
    [t_outUnits]  
)  
⇒ f_output/nil
```

Converts the string specifying a value and its units *f_output* to *t_outUnits* and returns it as *f_output*.

t_inUnits String giving the input number to convert and its units

t_outUnits String giving the new units for *f_output*. If *t_outUnits* is nil, converts the number to the current units of the layout.

Value Returned 2

f_output Converted value of *n_input*.

nil Failed to convert *n_input* due to incorrect argument(s).

Example 2

```
(axlMKSConvert ".5 MILS" "INCHES") -> 0.0005
```

axIMKSStr2UU

```
axIMKSStr2UU(  
    t_String  
)  
⇒ t_mksString/nil
```

Description

Converts an input string to a MKS string in current database units. If the input string is in MKS units, that is used as the basis for the conversion. If the string has no units, the function uses the default database units for the string.

Conversion may fail for the following reasons:

- Input string is not a legal MKS format.
- Conversion overflows the maximum database size allowed.

Notes:

- Conversion between metric and english units may result in rounding.
- Number returned is rounded to the database precision.

Argument

t_String Input string.

Value Returned

t_mksString MKS string in current database units.

nil Failed to convert input string. See earlier Description for possible reasons.

Example

```
axIMKSStr2UU( "100.1" ) -> "100.0 MILS"
```

Default conversion with the database in mils.

axlMapClassName

```
axlMapClassName(  
    t_oldName  
)  
⇒ t_newName
```

Description

Use this function to write a SKILL program that runs in both the Board and APD space. You can map from PCB Class names to the name appropriate to the program running your SKILL program. For example, you can write a program using PCB Class names and have the program run in APD.

The table shows the name mapping between PCB and APD.

Table 23-1 Class Name Mapping

PCB Class Name	APD Class Name
BOARD GEOMETRY	SUBSTRATE GEOMETRY
ETCH	CONDUCTOR
ANTI ETCH	ANTI CONDUCTOR
PACKAGE GEOMETRY	PART GEOMETRY
PACKAGE KEEPIN	PART KEEPIN
PACKAGE KEEPOUT	PART KEEPOUT
BOARD	SUBSTRATE

Argument

t_oldName PCB class name.

Value Returned

t_newName Appropriate class name based on product type.

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

Example 1

```
axlMapClassName( "ETCH" ) -> "CONDUCTOR"
```

Gets APD class name when in APD.

Example 2

```
axlMapClassName( "ETCH" ) -> "ETCH"
```

Gets PCB Editor class name when in PCB Editor.

axlMemSize

```
axlMemSize(  
)  
⇒ x_size
```

Description

Returns an estimate of memory use. Returns not what the program is *using*, rather what it is *requesting* from the operating system.

Argument

nothing

Value Returned

x_size	Estimate of memory use in bytes.
--------	----------------------------------

axlPPrint

```
axlPPrint(  
    t_name  
)  
⇒ t_pname
```

Description

Converts a string with PCB Editor's pretty print text function as follows:

- Ensures that the first character after a white space, _ (underscore) or / (forward slash) is capitalized.
- Ensures the rest of the text in the given string is lower case.

Argument

t_name A string.

Value Returned

t_pname The converted string.

Example

```
axlPPrint( "ETCH/TOP" ) -> "Etch/Top"
```

axIPdfView

```
axIPdfView(  
    t_pdfFile  
)  
⇒ t
```

Description

Displays a PDF file from PCB Editor. If just the filename is given, attempts to find the PDF file using PCB Editor's `PDFPATH` variable. Displays the PDF file in an external PDF viewer.

On UNIX, the only supported PDF viewer is Acroread. The program, Acroread, must be in your PATH.

On Windows, uses the default PDF viewer registered with the Windows registry. If there is no registered PDF viewer, the call fails.

Argument

t_pdfFile Name of PDF file to display.

Value Returned

t PDF file displayed.

nil Failed to display PDF file due one of the following:

 - No Acroread PDF viewer found on UNIX.

 - No PDF viewer registered on Windows.

Example

```
axIPdf( "allegro.pdf" )
```

axlRegExpls

```
axlRegexpIs(  
    t_exp  
)  
⇒ t/nil
```

Description

Determines whether an environment variable expression contains PCB Editor compatible wildcard characters. Certain select-by-name functions support wildcard characters. You can test for the presence of wildcards before calling the select-by-name type of functions.

Regular expressions used by PCB Editor are more compatible with the character set allowed in the PCB Editor object names than SKILL regular expressions. Do not use to test patterns being sent to the SKILL `regexp` family of functions.

Argument

t_exp SKILL symbol for the environment variable name.

Value Returned

t	Expression contains PCB Editor compatible wildcards.
nil	Expression does not contain PCB Editor compatible wildcards.

axlRunBatchDBProgram

```
axlRunBatchDBProgram(  
    t_prog  
    t_cmdFmt  
    [?logfile      t_logfile]  
    [?startMsg    t_startMsg]  
    [?reloadDB    t/nil]  
    [?noUnload    t/nil]  
    [?silent      t/nil]  
    [?noProgress  t/nil]  
)  
⇒ t/x_error
```

Description

Spawns batch jobs that require an open database via an abstract model. When the job completes, it prints a message and optionally reloads (?reloadDB) the database if successful. If the database is saved from the current active database, it uses a temporary name to avoid overwriting the database on disk.

Argument

<i>t_prog</i>	Name of the program to run.
<i>t_cmdFmt</i>	Command string.
<i>t_logFile</i>	Name of log file that the program creates. Registers this with the log file viewlog facility if the program ends in an error. If no log file is required, do not set this option. If no extension is given, adds .log as the extension.
<i>startMsg</i>	Enables a start message to display when the program starts. Defaults to the program name. If you override by providing this string, the message begins with "Starting..."
<i>reloadDB</i>	If you set this to t, the database reloads after a successful run of the program. If the batch program does not save the database, you need not reload the database.
<i>noUnload</i>	If you set this to t, you don't save the database to disk. You use this for a program that creates a new database or doesn't require an PCB Editor database. Default is nil.

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

<i>silent</i>	If you set this to <code>t</code> , no messages are displayed. Use when the user does not need to know that a program is spawned. Default is <code>nil</code> .
<i>noProgress</i>	If <code>t</code> , the progress meter does not display. Default is <code>nil</code> .
<i>noLogview</i>	When set, it prevents display if log file on program exit.
<i>noWarnOnExit</i>	When set, suppresses some exit warning messages.
<i>noExitMsgs</i>	When set, messages about program success or failure are suppressed.

Note: For `t_cmdFmt`, the formatting should include everything except the database file. Place a `%s` where the database should appear.

- To get a `%s` and still do a `sprintf` to fill in your optional command arguments, use a `"%%s"` as shown:

```
cmdFmt = "netrev -$ -q -r %s"
sprintf(cmdFmt, "%s -$ %s %s %%s", prog, argq argr)
```

Value Returned

<code>t</code>	Batch job ran.
<code>x_error</code>	Error number that is program dependent on failure.

Example 1

```
sprintf(format "genfeedformat -$ %s %%s", "-b")
axlRunBatchDBProgram("genfeedformat"
                     format
                     ?logfile "genfeed")
```

Spawns a `genfeedformat` which requires the database to be saved.

Example 2

```
axlRunBatchDBProgram("netin"
                     "netin -$ -g -y 1 netlist %s"
                     ?startMsg "Logic Import"
                     ?logfile "netin"
                     ?reloadDB t)
```

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

Spawns `netin`, a program that requires read/write to the database.

Note: ‘`-$`’ is a standard argument to PCB Editor batch programs that prevents prompting for missing input.

axlShowObject

```
axlShowObject(  
    lud_dbid  
)  
⇒ t/nil
```

Description

Displays the object data for each *dbid* in *lud_dbid* in a *Show Element* window. Does the same function as the interactive command `list element`. *lud_dbid* can be either a single *dbid* or a list of *dbids*.

Arguments

lud_dbid *dbid*, or list of *dbids*.

Value Returned

t	Displayed the Show Element window for any objects.
nil	Failed to display the Show Element window for any objects.

Example

```
axlDBCreatePropDictEntry(  
    "myprop", "real", list( "pins" "nets" "symbols"),  
    list( -50. 100), "level")  
axlClearSelSet()  
axlSetFindFilter(  
    ?enabled '("NOALL" "ALLTYPES" "NAMEFORM")  
    ?onButtons "ALLTYPES")  
axlSingleSelectName( "NET" "ENA2")  
axlDBAddProp(axlGetSelSet(), list("MYPROP" 23.5))  
axlShowObject(axlGetSelSet())  
⇒ t
```

1. Defines the string-valued property MYPROP.
2. Adds it to the net ENA2.
3. Displays the result to the user with `axlShowObject`.

axISleep

```
axISleep(  
    x_time  
)  
==> t/nil
```

Description

Sleeps specified time.

This is a replacement for Skill's sleep which is actually provided by the IPC Skill package. On Windows the IPC sleep crashes if you call axIEnterEvent.

If you are using the IPC interfaces then you should call axISleep instead of using sleep.

Arguments

x_time Time in seconds.

Value Returned

t All of the time.

axlSort

```
axlSort(  
    t_infile  
    t_outfile  
    [t_sortfields]  
    [t_sort_options]  
)  
⇒ t/nil
```

Description

Sorts contents of a given input file and places results in the output file. No warning is given if the output file overwrites an existing file - any checking must be done by the caller of this function.

Default sort is a left to right, ASCII ascending sort of the input file, with duplicate lines left in. You can control the sort behavior using the optional parameters.

Argument

<i>t_infile</i>	Name of the input file to sort.
<i>t_outfile</i>	Name of the file containing results of the sort.
<i>t_sortfields</i>	String specifying which fields to use as sort keys, the sort order of those fields, and how to interpret the data type of the field for sorting. String contains a series of triplets: field_number sortOrder fieldType String can contain multiple triplets separated by commas. Triplets can contain valid elements as shown:
<hr/>	
<hr/>	

Triplet Element	Description
<i>field_number</i>	Number representing the position of the field on the line, from left to right. Numbering starts at 1.
<i>sortOrder</i>	A - Ascending sort order D - Descending sort order
<i>fieldType</i>	A - Alpha I - Integer F - Float

An example of a *t_sortfields* triplet:

"3 A A, 5 D I, 1 A F"

This triplet sorts based on the following:

1. The third field, ascending, field type ASCII.
2. The fifth field, descending, field type Integer.
3. The first field, ascending, field type Float.

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

t_sort_options String containing directives controlling the global sort parameters. Sort options can appear in any order, and must be separated by commas. These options are available:

Option	Description
Field Delimiter	Supported delimiters include any character in punctuation character class except comma.
U	Remove duplicate lines. Default is keep duplicate lines.
D	Descending order. Default is ascending order.

An example of the *t_sort_options* string:

"! ,U"

This means to use the ‘!’ character as the field delimiter and to remove any duplicate lines.

Value Returned

- | | |
|-----|---|
| t | Sort successful. |
| nil | Sort unsuccessful due to incorrect arguments. |

Examples

Input file

```
2!3!4!5!6  
2!3!4!5!6  
5!6!7!8!9  
5!1!7!8!9  
2!2!3!4!5  
1!2!3!4!5  
3!4!5!6!7  
4!5!6!7!8
```

Example 1

```
(axlSort "input.txt" "output.txt" nil "!,U")
```

Results 1

```
1!2!3!4!5  
2!2!3!4!5  
2!3!4!5!6  
3!4!5!6!7  
4!5!6!7!8  
5!1!7!8!9  
5!6!7!8!9
```

Example 2

```
(axlSort "input.txt" "output.txt" "2 A I, 1 D I" "")
```

Results 2

```
5!1!7!8!9  
2!2!3!4!5  
1!2!3!4!5  
2!3!4!5!6  
2!3!4!5!6  
3!4!5!6!7  
4!5!6!7!8  
5!6!7!8!9
```

axlstrcmpAlpNum

```
axlstrcmpAlpNum(  
    t_str1  
    t_str2  
)  
⇒ t/nil
```

Description

Provides an alpha-numeric sort similar to `alphalessp` with one important distinction. If both strings end in the number, the number portion is separated and the two stripped strings are first compared. If they are equal, then the number sections are compared as numbers, rather than strings.

`alphalessp` sort U1 U10 U2

`axlstrcmpAlpNum` sort U1 U2 U10

Arguments

`t_str1` A string

`t_str2` A string

Value Returned

0 The two strings are equal.

+num If `t_str1` is greater than `t_str2` (1 goes after 2)

nil If `t_str1` is less than `t_str2` (1 goes before 2)

Example

```
l = '("U5" "U10" "U1" "U5" "U2")  
sort(l 'axlstrcmpAlpNum)  
====> ("U1" "U2" "U5" "U5" "U10")
```

axlVersion

```
axlVersion(  
    s_option  
)  
⇒ g_value/nil
```

Description

Returns PCB Editor or OS dependent data.

Argument

s_option SKILL symbol for the environment variable name.

Value Returned

Option	Value	Returns
none	<i>ls_values</i>	List of available options.
version	<i>f_value</i>	PCB Editor program version, for example, 15.1.
release	<i>t_value</i>	PCB Editor release value. Consists of a prefix and a number (< <i>p</i> >< <i>nn</i> >), where prefixes are as shown: A - Alpha B - Beta P - Production S - Patch For example, S23 indicates the 23rd patch release.
internalVersion	<i>t_value</i>	Internal program version, for example, v15-1-25A.
programName	<i>t_value</i>	Executable being run, for example, allegro.

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

Option	Value	Returns
displayName	<i>t_value</i>	Display name of program (may contain spaces.) Certain programs may change this during run-time depending on the license level.
formalName	<i>t_value</i>	Full formal name of program (may contain spaces). Certain programs may change during run-time depending on licensing level. The name can be the same or longer than <i>displayName</i> , and may change from release to release.
isWindows	<i>g_value</i>	<i>t</i> if Windows operating system. <i>nil</i> if UNIX operating system.
isSQ	<i>g_value</i>	<i>t</i> if Allegro SI, <i>nil</i> otherwise.
isPI	<i>g_value</i>	<i>t</i> if Allegro PCB PI Option 610, <i>nil</i> otherwise.
isSQPkg	<i>g_value</i>	<i>t</i> if Allegro Package SI 610, <i>nil</i> otherwise.
isSQPkg3D	<i>g_value</i>	<i>t</i> if Allegro Package SI 620, <i>nil</i> otherwise
isAllegroExpert	<i>g_value</i>	<i>t</i> if Allegro PCB Design 610 product, <i>nil</i> otherwise.
isAllegroDesigner	<i>g_value</i>	<i>t</i> if Allegro PCB Performance 210 or Allegro PCB Design 210 product, <i>nil</i> otherwise.
isAllegroPCB	<i>g_value</i>	<i>t</i> if Allegro PCB product, <i>nil</i> otherwise.
isAPD	<i>g_value</i>	<i>t</i> if Allegro Package Designer 610 product, <i>nil</i> otherwise.
isChipIOPlanner	<i>g_value</i>	CHECK TO SEE IF THIS IS STILL VALID
isSIPLayout	<i>g_value</i>	<i>t</i> if SIP Layout product, <i>nil</i> otherwise.
isSIPEngineer	<i>g_value</i>	<i>t</i> if SIP Engineer product, <i>nil</i> otherwise.
isSigxp	<i>g_value</i>	<i>t</i> if any version of SigXP, <i>nil</i> otherwise.
isSxExpert	<i>g_value</i>	<i>t</i> if SigXP Expert product, <i>nil</i> otherwise.
isSxExplorer	<i>g_value</i>	<i>t</i> if SigXP Explorer product, <i>nil</i> otherwise.

Allegro PCB and Package User Guide: SKILL Reference

Utility Functions

Example

```
axlVersion()
```

axlVersionIdGet

```
axlVersionIdGet()  
)  
⇒ x_time
```

Description

Returns an id stamp based upon computer time.

Argument

none

Value Returned

<i>x_time</i>	Returns an id stamp based on computer time.
---------------	---

axlVersionIdPrint

```
axlVersionIdPrintd(  
    x_time/t_time  
)  
⇒ t_printTime/nil
```

Description

Prints version_id.

VERSION_ID is stored as a property on the database root and on the symbol definitions as shown:

```
axlDBGetDesign() -> prop -> VERSION_ID
```

Use to determine if a symbol should be refreshed. VERSION_ID is updated every time the database is saved, except if done as part of an uprev.

Argument

<i>x_time/t_time</i>	VERSION_ID obtained from the database property or returned from axlVersionIdGet().
----------------------	--

Value Returned

<i>t_printTime</i>	Printable string in standard PCB Editor date/time format.
nil	Failed to print VERSION_ID due to incorrect argument.

Example

```
axlVersionIdPrint(axlDBGetDesign() -> prop -> VERSION_ID  
-> "Mon Dec 16 12:45:16 2004")
```

Math Utility Functions

Overview

This chapter describes the AXL-SKILL Math Utility functions.

axlDistance

```
axlDistance(  
    l_point1  
    l_point2  
)  
⇒ f_distance
```

Description

Returns the distance between two points. You may use floating point.

Arguments

l_point A point.

ll_line Two points at the ends of a line.

Value Returned

f_distance Distance between two points in floating point form.

Example

```
axlDistance(10:20 5:20) = 5.0
```

axlGeoEqual

```
axlGeoEqual(  
    f_one  
    f_two  
)  
⇒ t/nil
```

Description

Performs an equal comparison between two floating point numbers and determines if they are equal within plus or minus the current database accuracy.

Useful for comparing floating point numbers converted from strings (example - using `atof` function) with those obtained from an AXL database id. Since the conversion of these numbers takes different paths (string to float versus integer to float, in the case of the database) you can have different numbers.

Arguments

Two floating point numbers.

Value Returned

t	Given numbers are equal within the current database accuracy.
nil	Given numbers are not equal within the current database accuracy.

Example

```
axlGeoEqual(2.0 2.0)
```

axlGeoRotatePt

```
axlGeoRotatePt(  
    f_angle  
    l_xy  
    l_origin/nil  
    [mirror]  
) -> l_xyResult/nil
```

Description

Rotates *xy* about an origin by angle. Optionally applies mirror on the *x* axis.

Arguments

<i>f_angle</i>	Angle 0 to 360 degrees (support for 1/1000 of a degree); rotation is counter-clockwise for positive numbers.
<i>l_xy</i>	<i>xy</i> point to rotate in user units.
<i>l_origin</i>	Origin to rotate about; if <i>nil</i> uses (0, 0) mirror: optional mirror flag (<i>t</i> perform mirror).

Value Returned

<i>l_xyResult</i>	Rotated result.
<i>nil</i>	Error in arguments or rotation results in return being outside of the database extents.

Examples

Rotate:

```
axlGeoRotatePt(45.0 10:200 5:2) -> (-131.4716 145.5427)
```

Rotate and mirror:

```
axlGeoRotatePt(45.0 10:200 5:2 t) -> (-138.5427 138.4716)
```

Rotate about 0,0:

```
axlGeoRotatePt(90.0, 100:0 nil) -> (0.0 100.0)
```

Allegro PCB and Package User Guide: SKILL Reference

Math Utility Functions

axlIsPointInsideBox

```
axlIsPointInsideBox(  
    l_point  
    l_box  
)  
⇒ t/nil
```

Description

Returns *t* if a point is inside or on the edge of a box. Also see [axlGeoPointInShape](#) on page 885 for *dbid*-based tests. You may use floating point.

Arguments

<i>l_point</i>	A point.
<i>l_box</i>	A bounding box.

Value Returned

<i>t</i>	Point is inside or on edge of box
<i>nil</i>	Point is outside of box.

Example

```
axlIsPointInsideBox(10:20 list(5:20 15:30)) = t  
axlIsPointInsideBox(0:20 list(5:20 15:30)) = nil  
axlIsPointInsideBox(15:20 list(5:20 15:30)) = t
```

axlIsPointOnLine

```
axlIsPointOnLine(  
    l_point  
    ll_line  
)  
⇒ t/nil
```

Description

Returns `t` if point is on a given line or `nil` if not on the line. You may use floating point.

Arguments

<code>l_point</code>	A point.
<code>ll_line</code>	Two end points.

Value Returned

<code>t</code>	Point is on the specified line.
<code>nil</code>	Point is not on the specified line.

Example

```
axlIsPointOnLine(10:20 list(5:20 15:30)) = nil  
axlIsPointOnLine(15:20 list(5:20 15:30)) = nil
```

axlLineSlope

```
axlLineSlope(  
    ll_line  
)  
⇒ t_slope/nil
```

Description

Returns the slope of a line. You may use floating point.

Arguments

<i>ll_line</i>	Two end points.
----------------	-----------------

Value Returned

<i>t_slope</i>	Slope of the line.
----------------	--------------------

<i>nil</i>	Line is vertical.
------------	-------------------

Example

```
axlLineSlope(list(5.0:20.10 15.4:30.2)) = 0.9711538  
axlLineSlope(list(5:20 5:40)) = nil
```

axlLineXLine

```
axlLineXLine(  
    l_seg1  
    l_seg2  
)  
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

t	Returns t always.
---	-------------------

axlMPythag

```
axlMPythag(  
    l_pt1  
    l_pt2  
) -> f_distance/nil
```

Description

Calculates distance between two points using pythagoras. This is faster then building this code in Skill.

The *l_ptN* is an x:y coordinate.

Arguments

l_pt1, *l_pt2* Two xy points.

Value Returned

f_distance Distance between points.

nil Arguments were not xy points.

See also

`axlMXYAdd`

Example

```
axlMPythag(1263.0:1062.0 1338.0:1137.0) -> 106.066
```

axlMXYAdd

```
axlMXYAdd(  
    l_pt1  
    l_pt2  
) -> l_pt/nil
```

Description

This does a `l_pt1 + l_pt2` and returns the result.

The `l_ptN` is an x:y coordinate.

Arguments

`l_pt1, l_pt2` Two xy points.

Value Returned

`l_pt` Returns coordinate that is result of addition.

`nil` Arguments are not coordinates.

See also

`axlMXYSub`

Example

```
axlMXYAdd(1263.0:1063.0 1338.0:1137.0) -> (2601.0 2200.0)
```

axlMXYSub

```
axlMXYSub(  
    l_pt1  
    l_pt2  
) -> l_pt/nil
```

Description

This does a `l_pt1 - l_pt2` and returns the result.

The `l_ptN` is an x:y coordinate.

Arguments

`l_pt1, l_pt2` Two xy points.

Value Returned

`l_pt` Returns coordinate that is result of subtraction.

`nil` Arguments are not coordinates.

See also

`axlMXYAdd`

Example

```
axlMXYSub(' (1263.0 1063.0) '(1338.0 1137.0)) -> (-75.0 -74.0)
```

axl.ol.ol2

```
axl.ol.ol2(  
    l_seg1  
    l_seg2  
)  
⇒ l_result
```

Description

Finds the intersection point of two lines. If the lines intersect, returns the intersection point with a distance of 0. If the lines do not intersect, the distance is not zero and the function returns t.

Arguments

l_seg1 1st line segment (list *x1:y1 x2:y2*)

l_seg2 2nd line segment (list *x1:y1 x2:y2*)

Value Returned

nil Error due to incorrect argument.

(car l_result) Intersect or nearest point on seg1

(cadr l_result) Intersect or nearest point on seg2

(caddr l_result) Distance between the two intersect points.

Examples

Data for examples

```
a=list(1:5 5:5)
b=list(2:5 4:2)
c=list(0:0 5:0)
d=list(4:5 7:5)
```

Example 1

```
axl.ol.ol2(a b)
⇒ ((2.0 5.0) (2.0 5.0) 0.0)
```

Intersects line, returns intersection point, note distance of 0.

Example 2

```
axl.ol.ol2(a c)
⇒ ((3.0 5.0) (3.0 0.0) 0.0)
```

Lines don't intersect, returns closest point on each line and distance.

Example 3

```
axl.ol.ol2(a d)
⇒ ((4.5 5.0) (4.5 5.0) 0.0)
```

Lines overlap, distance of 0 and selects mid-point of the overlap.

Allegro PCB and Package User Guide: SKILL Reference

Math Utility Functions

Database Miscellaneous Functions

Overview

This chapter describes the AXL-SKILL functions that don't fit into other sections.

axlAirGap

```
axlAirGap(
    o_item1DBID
    o_item2DBID
    [t_layer]/nil
    [s_full]
)
==> l_airGapData/nil/(s_error l_airGapData/l_errorData)
```

Description

Finds the air gap and location between two given items. Gap is the same as reported by the `show measure` command. The set of legal objects is: pins, vias, shapes, cline segs, other segs, DRC errors and text. When doing pad comparison, you provide the layer, or PCB Editor uses the current active layer. The layer syntax should either be `etch/<subclass>` or `<subclass>`.

Output data appears in one of the following formats depending on the `s_full` option:

- Default is `s_full` (`s_full==nil`) returns the `l_airGapData` or a `nil` if there is an error. If `s_full` is `t` then data is returned as `(s_error l_airGapData)` where `s_error` is one of the following:

<code>t</code>	Success (<code>t (l_airGapData)</code>)
<code>'NOMATCH</code>	No subclass matches between pin or via and object. Returns object's layer. (<code>NOMATCH (t_layer)</code>)
<code>'RANGE</code>	No subclass match between two etch elements (one or both must be a pad element (pin or via). If common layers exist, PCB Editor returns the top and bottom layer where matches exist otherwise returns <code>nil</code> : (<code>ETCH (t_topMatch t_bottomMatch)</code>)
<code>'INVALID</code>	One or both elements are invalid. Data return format: (<code>INVALID nil</code>). For distance between a pad and non-pad element, use the element subclass, if the non pad element is on etch. If not, use the <code>t_layer</code> which defaults to the active layer, if undefined. If non-etch and the pad are undefined on the subclass, determine the subclasses on which the pad is defined and return those layers (<code>xxx</code>). For distance between two pads, return gap based upon the active etch subclass, if <code>t_layer</code> is <code>nil</code> . Otherwise use

t_layer to determine gap. If one or both pads do not exist on that layer, there is an error.

Arguments

<i>o_item1DBID</i>	dbid of the first item.
<i>o_item2DBID</i>	dbid of the second item.
<i>t_layer</i>	Optional layer used to resolve gap comparison between two pin or via elements.
<i>s_full</i>	Return additional info to clarify error.

Value Returned

<i>l_airGapData</i>	List containing the following items: (<i>l_airGapPt1</i> <i>l_airGapPt2</i> <i>f_airGapDistance</i>)
where:	
<i>l_airGapPt1</i>	(X,Y) point on the first item where the air gap is measured.
<i>l_airGapPt2</i>	(X,Y) point on the second item where the air gap is measured.
<i>f_airGapDistance</i>	Distance between the two points.
<i>nil</i>	Input data error; element 1 and 2 are the same or no air gap can be computed between the two items. If <i>t_layer</i> is used but does not specify an etch layer.
<i>s_error</i>	See error symbols listed above.

Examples

Basic input:

```
axlAirGap(e11 e12)
-> ((1337.5 1100.0) (1362.5 1100.0) 25.0)
```

Basic input layer:

```
axlAirGap(e11 e12 "TOP")
-> ((1337.5 1100.0) (1362.5 1100.0) 25.0)
```

Allegro PCB and Package User Guide: SKILL Reference

Database Miscellaneous Functions

Full output success:

```
axlAirGap(el1 el2 nil t)
-> (t ((1337.5 1100.0) (1362.5 1100.0) 25.0))
```

Full output failure:

```
axlAirGap(el3 el2 nil t)
-> (RANGE ("TOP" "GND"))
```

axlExtentDB

```
axlExtentDB(  
)  
⇒ l_bBox/nil
```

Description

Determines a design type and returns the *bBox* extent. See [axlExtentLayout](#) and [axlExtentSymbol](#) for what PCB Editor considers an extent.

Arguments

None

Values Returned

<i>l_bBox</i>	Returns <i>bBox</i> extent.
<i>nil</i>	Unknown drawing type.

axlExtentLayout

```
axlExtentLayout(  
)  
⇒ l_bBox/nil
```

Description

Computes the layout extents and returns the smallest bounding box to be used for window-fit. Only lines, linesegs, and shapes are searched on selected layers in the following order:

- 1.** BOARD GEOMETRY/OUTLINE
- 2.** PACKAGE KEEPIN/ALL
- 3.** ROUTE KEEPIN/ALL

The first layer with any elements is used to determine the layout extents. If no elements are found on these layers, the design extents are returned.

Arguments

None

Value Returned

<i>l_bBox</i>	Returns <i>bBox</i> of the layout.
nil	Error such as the failure of <code>axlVisibleGet</code> .

axlExtentSymbol

```
axlExtentSymbol(  
)  
⇒ l_bBox
```

Description

Computes the bounding box enclosing all objects visible for a drawing (a .dra file).

Arguments

None

Value Returned

<i>l_bBox</i>	Smallest bounding box enclosing all visible objects. If no objects are visible, set to the design extents.
---------------	--

axIDRCGetCount

```
axlGetDRCCount  
)  
⇒ x_count
```

Description

Returns the total number of DRCs in the design. Note the design DRC may be out of date.

Arguments

None

Value Returned

x_count DRC count.

axlGeoPointInShape

```
axlGeoPointInShape(  
    l_point  
    o_dbid  
    [t/nil]  
)  
⇒ t/nil
```

Description

Given a point and a shape *dbid*, determines whether that point is inside or outside the shape. For a shape with voids, a point is considered *outside* the given shape if inside a void.

Arguments

<i>l_point</i>	Point to check.
<i>o_dbid</i>	Shape to check.
[t/nil]	t means include voids, nil means use the shape outline only. Default is t.

Value Returned

t	Point is inside the shape.
nil	Point is outside the shape, or incorrect arguments were given.

axlIsHighlighted

```
axlIsHighlighted (
    o_dbid
)
==> x_highlightColor/nil
```

Description

If the object is permanently highlighted returns the highlight color; otherwise nil.

Note: Pins can be highlighted.

Only symbols, nets, pins and DRC errors can be highlighted. Cadence suggests that you do not highlight drc objects unless they are external DRCs, since PCB Editor DRCs are frequently recreated.

Arguments

o_dbid A dbid for which highlighting information is desired.

Value Returned

x_highlightColor Highlight color; nil if not highlighted, or object does not support highlighting.

See also [axlHighlightObject](#).

Examples

See [axlHighlightObject](#).

axlTestPoint

```
axlTestPoint  
  o_dbid  
  top|bottom|nil  
)  
⇒ t/nil/s_error
```

Description

Sets or clears a pin and/or via's test point status. Abides by the rules of the testprep parameter form in its ability to add a test point (see possible errors, below). If testprep rules prevent adding a test point, an error symbol is returned. If the command fails for other reasons, nil is returned. On success, a t is returned.

If you add a test point to a pin/via that already has a test point, the existing test point is replaced.

Not enabled in a symbol editor.

Adds test point text using same rules as the testpoint manual command.

Note: Does not delete associated test point text. Use axlDeleteObject and axlDBGetAttachedText.

Arguments

<i>o_dbid</i>	Pin or via dbid
<i>g_mode</i>	Add test point to top or bottom, or clear one.

Value Returned

<i>t</i>	Object changed.
<i>nil</i>	Error other than test point checks.
<i>s_error</i>	Symbol indicating an error from testprep parameter check.

Errors

PAD_TOO_SMALL	Size does not meet parameter minimums
PAD_UNDER_COMP	Padstack under component
PIN_OFF_GRID	Pin off grid
PAD_UNDEFINED	Layer of padstack not defined on required layer
PAD_NOT_SMD	Padstack must be a SMD
PAD_NOT_THRU	Padstack must be a thru pad
PAD_IN_NO_PROBE_AREA	Testpoint pad in NO_PROBE area
PIN_IS_VIA	Pin type requires a via or any point
PIN_NOT_VIA	Pin type requires a via
PIN_NOT_OUTPUT	Pin type requires an output pin for test point
PIN_NOT_IO	Pin type requires an IOpin for test point
PIN_TOO_CLOSE	Pin too close to another test point
PAD_UNDER_PIN	Test point under another pin
PIN_NOT_NODE	Test point requires a node for testbench
FIXED_TEST_POINTS	Testpoints are fixed and cannot be removed
OTHER	Unclassified error

axlChangeNet

```
axlChangeNet(  
    o_dbid  
    t_netname  
)  
⇒ t/nil
```

Description

Changes the net an object is currently on. Restricted to shapes and filled rectangles (frectangles). Returns t when successful.

Failure can occur for the following reasons:

- Object is not a shape or frectangle on class.
- Netname does not exist.

A potential side effect of this function is that it may not properly reconnect two touching connect-line segments previously connected by the shape.

Arguments

o_dbid Shape *dbid*

t_netname Name of a net

Value Returned

t Object changed.

nil No object changed.

axlSegDelayAndZ0

```
axlSegDelayAndZ0(  
    o_clineSegDbid  
)  
⇒ (f_delay f_z0)/nil
```

Description

Returns the delay and impedance of a cline segment. Returns `nil` if a segment isn't a cline segment. Normally, delay is in nanoseconds and impedance is in ohms.

This function is noisy if you pass in non-cline segments.

Arguments

`o_clineSegDbid` Segment cline

Value Returned

`f_delay f_z0` Delay and impedance in current MKS units.

`nil` Segment is not a cline segment.

Logic Access Functions

Overview

This chapter describes the AXL-SKILL functions related to schematic capture or the logical definition of the design.

axlDBCreateConceptComponent

```
axlDBCreateConceptComponent(  
    s_refdes  
    s_partPath  
    s_logName  
    s_primName  
    [s_pptRowName]  
)  
⇒ r_dbid/nil
```

Description

Given the Concept information needed to describe an PCB Editor device, create the PCB Editor component and return its *dbid*. If the component definition already exists, use the existing definition to create the new component. If the component definition does not exist, create a new definition and then create the component instance. Concept information comes from a `chips_ptr` file and from a physical parts table (PPT). Determine the location of this information using the `cptListXXX` family of routines, which allow browsing through a Concept library.

Arguments

<code>s_refDes</code>	Reference designator for the new component.
<code>s_partPath</code>	Full path to the <code>chips_ptr</code> file containing the description of the desired logical part. Determine using the <code>cptListComponentLibraries</code> function.
<code>s_logName</code>	Name of the desired logical part. You can determine this using the <code>cptListComponentPrimitives</code> function.
<code>s_primName</code>	Name of the desired primitive. You can determine this using the <code>cptListComponentPrimitives</code> function.
<code>s_pptRowName</code>	Name of the desired PPT part. Concept data may not include specific device information which would be contained in a PPT. Indicates a specific row in a PPT from which to create the component. You can determine this using the <code>cptListComponentDevices()</code> function.

Value Returned

r_dbid *dbid* of the new PCB Editor component instance.

nil Unable to create component instance.

Note: The actual name of the resulting PCB Editor device is generated automatically. If using the `cptListComponentDevices()` function, then the name should have been created already and is included in the return values of that function. Name is determined by the Concept library data you use to create the part.

This function is meant to be called from SKILL.

axlDBCreateComponent

```
axlDBCreateConceptComponent (
    s_refDes
    s_deviceName
    [ s_package ]
    [ s_value ]
    [ s_tolerance ]
)
==> r_dbid/nil
```

Description

Given the information needed to describe a PCB Editor device, create the PCB Editor component and return its `dbid`. If the component definition already exists, use the existing definition to create the new component. If the component definition does not exist, create a new definition using the device file and create the new component.

Arguments

<code>s_refDes</code>	The reference designator for the new component.
<code>s_deviceName</code>	Name of PCB Editor device file.
<code>s_package</code>	Package name to be used for the component. This overrides the value found in the device file (can be nil).
<code>s_value</code>	"Value" attribute value. This will override the value found in the device file (can be nil).
<code>s_tolerance</code>	"Tolerance" attribute value. This will override the value found in the device file (can be nil).

Value Returned

<code>r_dbid</code>	<code>dbid</code> of new PCB Editor component.
<code>nil</code>	If unable to create component.

Note: If you change an existing component definition by specifying a new value for package, value, or tolerance, you need a device file.

axlDBCreateManyModuleInstances

```
axlDBCreateManyModuleInstances(
    t_name
    t_moddefName
    x_tileStartNum
    l_origin
    l_offset
    x_num_tiles
    f_rotation
    x_logicMethod
    [l_netExcept]
)
==> o_result/nil
```

Description

Creates multiple module instances in the design. By reducing the number of times the module definition file opens, this function optimizes performance when creating several instances of the same module.

Arguments

<i>t_name</i>	Prefix of the names of the module instances.
<i>t_moddefName</i>	Name of the module definition/
<i>x_tileStartNum</i>	Tile numbering start number and increment by 1 for each tile.
<i>l_origin</i>	First location of first module.
<i>l_offset</i>	List containing the offset wanted between module origins.
<i>x_numTiles</i>	The number of module instances to be place.
<i>f_rotation</i>	Angle of rotation for the module instance.
<i>x_logicMethod</i>	Flag to indicate where the logic for the module comes from.
0	No logic.
1	Logic from schematic.

2

Logic from module definition.

l_netExcept

Optional list of net names to add to the net exception list.

Value Returned

lo_result If successful, returns the list of database objects that belong to the module instances created.

nil

Creation of module instance could not be completed.

See also `axlDBCreateModuleDef`, `axlDBCreateModuleInstance`

Example

Add five module instances based on the module definition file `mod.mdd`, starting at point `(10,10)` and offsetting by `(5,0)` every time:

```
modinsts = axlDBCreateManyModuleInstances(  
    "Num" "mod" 2 10:10 '(5 0) 5 0 2 '("GND" "+5"))
```

Creates five module instances named `Num2`, `Num3`, `Num4`, `Num5`, `Num6`.

axlDBCreateModuleDef

```
axlDBCreateModuleDef(  
    t_name  
    l_origin  
    l_objects  
)  
⇒ t/nil
```

Description

Creates a module based on existing database objects.

Arguments

<i>t_name</i>	String providing the name of the module definition. File name is the module definition name appended with .mdd.
<i>l_origin</i>	Coordinate serving as the origin of the module definition.
<i>l_objects</i>	List of objects to add to the module.

Value Returned

<i>t</i>	Module definition successfully created.
<i>nil</i>	No module definition created.

Example

```
axlSetFindFilter(?enabled '("noall" "components") ?onButtons '("noall"  
    "components"))  
axlSingleSelectName("COMPONENT" "U1")  
comp1 = car(axlGetSelSet())  
axlSingleSelectName("COMPONENT" "U2")  
comp2 = car(axlGetSelSet())  
axlDBCreateModuleDef("comps" '(0 0) '(comp1 comp2))  
⇒ A module definition file named comps.mdd is created.
```

Creates a module definition file containing two components.

axlDBCreateModuleInstance

```
axlDBCreateModuleInstance(  
    t_name  
    t_moddef_name  
    l_origin  
    r_rotation  
    i_logic_method  
    l_net_except  
)  
⇒ o_result/nil
```

Description

Allows you to use or place a previously defined module.

Arguments

<i>t_name</i>	String providing name of the module instance.
<i>t_moddef_name</i>	String providing name of the module definition to base the instance on.
<i>l_origin</i>	Coordinate location to place the origin of the module definition.
<i>r_rotation</i>	Angle of rotation for the module instance.
<i>i_logic_method</i>	Flag indicating where the logic for the module comes from: 0 - no logic 1 - logic from schematic 2 - logic from module definition.
<i>l_net_except</i>	Optional list of net names to add to net exception list.

Value Returned

<i>o_result</i>	Database object that is the group used to represent the module instance.
nil	Module instance not created.

Example

```
modinst = axlDBCreateModuleInstance("inst" "mod" '(500 1500) 2 '(("GND" "+5"))
```

Adds a module instance based on the module definition file mod.mdd.

axlDBCreateSymDefSkeleton

```
axlDBCreateSymDefSkeleton(  
    l_symbolData  
    l_extents  
    [l_pinData]  
)  
==> l_result/nil
```

Description

Creates a minimal symbol definition. While the symbol name and type must be provided, the instance is created only with pins. Once this definition has been created, you add the rest of the symbol geometry with additional axlDBCreate calls. This provides the ability to create symbols that do not exist in the library.

Arguments

<i>l_symbolData</i>	A list of (<i>t_symbolName</i> [<i>t_symbolType</i>]).
<i>t_symbolName</i>	Name of the symbol.
<i>t_symbolType</i> "	Package (default), mechanical, or format.
<i>l_extents</i>	The lower left and upper right corners of the symbol def extents.
<i>l_pinData</i>	List of <i>axlPinData</i> defstructs for the pins.

Value Returned

axlDBCreateSymDefSkeleton nil if not created, or *axlDBID* of the symbol definition.

axIDbidName

```
axlDbidName(          o_dbid  
    ) -> t_name/nil
```

Description

Provides the standard PCB Editor name of a database object. Many of the named PCB Editor objects (for example, nets) have names defined in the name attribute (for example, dbid->name) but other objects (for example, clines and pins) either do not have the desired reporting name or are unnamed.

Arguments

o dbid A PCB Editor database id.

Note: Some PCB Editor database ids are pseudo ids (for example, ax1DBGetDesign and pads) and generate a nil return.

Value Returned

PCB Editor name of object, or `nil` if not a `dbid` or true PCB Editor database object.

Examples

This uses the `ashOne` selection function found in:

```
<cdsroot>/share pcb/examples/skill/examples/ash-fxf/ashone.il
```

Pin name:

```
pin = ashOne()  
ax1DbidName(pin)  
-> "U1.1"
```

Cline name:

```
cline = ashOne()
ax1DbidName(cline)
-> "Net3. Etch/Top"
```

axlDiffPair

■ Add DiffPair

```
axlDiffPair(  
    t_diffpair  
    o_net1/t_net1  
    o_net2/t_net2  
)  
⇒ o_diffpair/nil
```

■ Modify DiffPair

```
axlDiffPair(  
    o_diffpair/t_diffpair  
    o_net1/t_net1  
    o_net2/t_net2  
)  
⇒ o_diffpair/nil
```

■ Delete DiffPair

```
axlDiffPair(  
    o_diffpair/t_diffpair  
)  
⇒ t/nil
```

Description

Creates, modifies, or deletes a differential pair. In all cases you can pass names or a *dbid*.

Note: If the differential pair was created due to Signoise models, you cannot modify or delete it. Consequently, you cannot modify or delete the differential pair if the following is true:

```
diffpair_dbid->prop->DIFFP_ELECTRICAL ==t.
```

Arguments

o_diffpair Diffpair *dbid*

t_diffpair Diffpair name.

o_net Net *dbid*.

t_net Net name.

Value Returned

Values returned depend on whether adding, modifying, or deleting.

o_diffpair *dbid* of new or modified diffpair

t Diffpair deleted.

nil Error due to incorrect arguments.

Example 1

```
DPdbid = axlDiffPair( "DP" "NET1+" "NET2-" )
```

Creates a differential pair and names it DP1.

Example 2

```
DPdbid = axlDiffPair(DPdbid "NET1+" "NET1-" )
```

Modifies a differential pair.

Example 3

```
axlDiffPair(DPdbid)
```

Deletes a differential pair.

Example 4

```
axlDiffPair(axlDBGetDesign()->diffpair)
```

Delete all differential pairs in design.

axlDiffPairAuto

```
axlDiffPairAuto(  
    t_diffPairPrefix  
    t_posNetPostfix  
    t_negNetPostfix  
    [g_returnDiffPairList]  
)  
⇒ x_cnt/(xcnt lo_diffpair)/nil
```

Description

Allows automatic generation of the diffpair. Generates the set of diffpairs based on the provided positive (*t_posNetPostfix*) and negative (*t_negNetPostfix*) postfixes used in your net naming.

You may provide a prefix (*t_diffPairPrefix*) used in generating the diffpair names of the form: <*t_diffPairPrefix*> + *netname* - *postfix*.

If nets are part of busses and end the bitfield syntax (<1>), the syntax portion is ignored when performing suffix matching. If a diffpair is created the bit number is added to the base net name used in forming a diffpair. For example, given two nets; DATA_P<1> and DATA_N<1> will result in a diffpair called DP_DATA1 with this call:

```
axlDiffPairAuto( "DP_ " "_P" " _N" )
```

Arguments

<i>t_diffPairPrefix</i>	String to prefix diffpair names. Use " " if no prefix is desired.
<i>t_posNetPostfix</i>	Postfixes used to identify + diffpair members.
<i>t_negNetPostfix</i>	Postfixes used to identify - diffpair members.
<i>g_returnDiffPairList</i>	Controls return.

Value Returned

Returns depend on value of *g_returnDiffPairList* as shown:

g_returnDiffPairList	Value Returned	Description
nil	<i>x_cnt</i>	Number of diffpairs created.
t	(<i>x_cnt</i> , <i>lo_diffpair</i>)	List of diffpairs created

Examples

Example nets

Two nets called NET1+ and NET1- are passed to this function.

Example 1

```
axlDiffPairAuto( "DP_ " "+ " "- " )
```

Shows diffpair creation and name generation. Results in one diffpair called DP_NET1 with members NET1+ and NET1-.

Example 2

```
axlDiffPairAuto( " " "+ " "- " )
```

Gives the same result as the previous example, but names the diffpair NET1.

axlDiffPairDBID

```
axlDiffPairDBID(  
    t_name  
)  
⇒ o_dbid/nil
```

Description

Returns the *dbid* of the named diffpair (*t_name*) if it exists in the database.

Arguments

<i>t_name</i>	Diffpair name.
---------------	----------------

Value Returned

<i>o_dbid</i>	<i>dbid</i> of diffpair if it exists.
---------------	---------------------------------------

<i>nil</i>	Diffpair does not exist.
------------	--------------------------

axlDBGetExtents

```
axlDBGetExtents(o_dbid
                  g_visibleOnly
                )
==> bBox/nil
```

Description

Provides the extents of a physical database object. You choose between getting the extents of the entire object (even if it is currently not visible) or just the current visible objects.

With the visible option the extents may be smaller than the `bBox` attribute of the element `dbid bBox`.

Arguments

`o_dbid` dbid of an element.

`visible_only` t return the extents of visible parts of the element.

`nil` Return the extents of the entire element.

Value Returned

`bBox` The extents of the element. This might be zero extents, if the object has no extents visible or does not have extents (for example, nets).

`nil` Error; bad arguments

axlMatchGroupAdd

```
axlMatchGroupAdd(  
    o_mgdbid/t_mgName  
    o_dbid/lo_dbid  
) ==> t/nil
```

Description

Adds members to a matched group. Eligible members are:

- nets (if a net is part of an xnet the xnet is added to the group)
- xnets
- pinpairs

See discussion in `axlDBMatchGroupCreate`.

Command fails in product tiers that do not support electrical constraints or the symbol editor.



Tip
Using dbids is faster than using names.

Arguments

o_mgdbid dbid of a match group.

t_mgName Name of a match group.

o_dbid Legal database dbid to add to match group.

lo_dbid List of legal database dbids to add to match group.

Value Returned

t Added elements.

nil Failed one or more element additions.

See also `axlMatchGroupCreate`

Example

Add two nets to match group created in axlMatchGroupCreate:

```
mg = car(axlSelectByName( "MATCH_GROUP" "MG1" ))  
nets = axlSelectByName( "NET" '( "B1_OUT" "B2_OUT" ))  
axlMatchGroupAdd(mg nets)
```

axlMatchGroupCreate

```
axlMatchGroupCreate(  
    t_name  
) ==> o_mgdbid
```

Description

Creates a new match group. If a match group already exists with the same name, `nil` is returned. Match groups need to be populated, or they are deleted when saving. Command fails in product tiers that do not support electrical constraints or the symbol editor.

If the match group was partially or completely created from an ECset, you can delete it, but it reappears when ECset flattening is required due to modifications in the design. SKILL functions do not indicate ECset-derived match groups.

RELATIVE_PROPAGATION_DELAY

The `RELATIVE_PROPAGATION_DELAY` property can be added to the match group, xnet and pinpairs. If you add it to the match group then any match group member that does not have the property inherits it from the match group. Match groups contain the following elements: xnets, nets, and pinpairs. If a net is part of an xnet, the xnet is added to the match group. Xnets and pinpairs can belong to multiple match groups, so an RPD property exists on the dbid for nets, xnets and pinpairs. This property is a list of lists where each sub-list contains a match group dbid and the `RELATIVE_PROPAGATION_DELAY` value:

```
rpd = ( (o_mgDbid t_rpdValue) . . . )
```

If a pinpair belongs to multiple match groups, you see two lists. In cases where the dbid belongs to a match group but has no `rpd` value, the `t_rpdValue` reports a `nil`.

You can add and delete properties to a match group or pinpair dbid using `axlMatchGroupProp`. When creating the property value, you must include the match group name but not an explicit pinpair. For example, the following adds an RPD property to a match group named MG2:

```
axlDBAddProp(mg ' ("RELATIVE_PROPAGATION_DELAY" "MG2:G:::0 ns:5 %") )
```

Additional restrictions for this property are:

- Pinpairs should leave the pinpair section empty ("::").

Example: "MG2:G:::0 ns:5 %"

- Match groups the pinpair not reference an explicit pinpair but, if not empty should be general pinpairs (for example, "AD:AR", "D:R" etc.)

- If nets and xnets are part of multiple Match Groups, they appear concatenated in the RELATIVE_PROPAGATION_DELAY property. Any pinpairs that are part of the net appear as part of the property at the net or xnet level. The RPD property that is dbids for pinpairs, net and xnets breaks the concatenation. Use `axlMatchGroupProp` for modification of the RELATIVE_PROPAGATION_DELAY property for all objects.

Arguments

t_name Name of match group (changed to upper case).

Value Returned

nil Error or match group with same name already exists.

o_mgdbid: dbid of match group.

See also `axlPinPairSeek`, `axlPinsOfNet`, `axlMatchGroupCreate`, `axlMatchGroupDelete`, `axlMatchGroupAdd`, `axlMatchGroupRemove`, `axlMatchGroupProp`.

Example

Create a match group called MG1:

```
mg = axlMatchGroupCreate( "mg1" )
```

axlMatchGroupDelete

```
axlMatchGroupDelete(  
    o_mgdbid/t_mgName  
) -> t/nil
```

Description

This deletes a match group. The command fails in product tiers that do not support electrical constraints or the symbol editor.



Tip
Using dbids is faster than using names.

Arguments

o_mgdbid dbid of a match group.

t_mgName Name of a match group.

Value Returned

t Match group deleted.

nil Failed.

See also `axlMatchGroupCreate`.

Examples

Delete match group created in `axlMatchGroupCreate`:

```
mg = car(axlSelectByName( "MATCH_GROUP" "MG1" ))  
axlMatchGroupDelete(mg)
```

or

```
axlMatchGroupDelete( "MG1" )
```

axlMatchGroupProp

```
axlMatchGroupProp(  
    o_mgdbid/t_mgName  
    o_dbid  
    t_value/nil  
)==> t/nil
```

Description

Adds or removes the RELATIVE_PROPAGATION_DELAY property from a member of a match group. Property must be a legal RPD syntax that includes the RPD name.

The command fails in product tiers that do not support electrical constraints or the symbol editor.

See discussion in axlDBMatchGroupCreate.



Using dbids is faster than using names.

Arguments

<i>o_mgdbid</i>	dbid of a match group
<i>t_mgName</i>	Name of a match group
<i>o_dbid</i>	Legal database dbid to of a member of the match group
<i>t_value</i>	RELATIVE_PROPAGATION_DELAY value in legal syntax. If value is nil; removes the property.

Value Returned

<i>t</i>	Added elements.
nil	Failed one or more element additions.

See also axlMatchGroupCreate.

Examples

Add two nets to match group created in axlMatchGroupCreate:

```
mg = car(axlSelectByName( "MATCH_GROUP" "MG1" ))
nets = axlSelectByName( "NET" '( "B1_OUT" "B2_OUT" ) )
n1 = car(nets)
n2 = cadr(nets)
axlMatchGroupAdd(mg nets)
```

Add properties:

```
axlMatchGroupProp(mg n1 "MG1:G:::100 ns:5 %")
axlMatchGroupProp(mg n2 "MG1:G:AD:AR:0 ns:5 %")
```

Remove property from n2:

```
axlMatchGroupProp(mg n2 nil
```

axlMatchGroupRemove

```
axlMatchGroupRemove(  
    o_mgdbid/t_mgName  
    o_dbid/lo_dbid  
) ==> t/nil
```

Description

Removes elements from an existing match group. Elements must be members (attribute groupMembers) of the match group.

The command fails in product tiers that do not support electrical constraints or the symbol editor.



Tip
Using dbids is faster than using names.

Arguments

<i>o_mgdbid</i>	dbid of a match group.
<i>t_mgName</i>	Name of a match group.
<i>o_dbid</i>	Legal database dbid to remove from match group.
<i>lo_dbid</i>	List of legal database dbids to remove from match group.

Value Returned

<i>t</i>	Removed elements.
<i>nil</i>	Failed one or more element removals.

See also `axlMatchGroupCreate`.

Example

To match group in example `axlMatchGroupAdd` remove one of the nets:

```
axlMatchGroupRemove(mg car(nets))
```

axlPinPair

Add

```
axlPinPair(  
    o_pin1/t_pin1  
    o_pin2/t_pin2  
)==> o_pinpair
```

Delete

```
axlPinPair(  
    o_pinpair/lo_pinpair  
)==> t/nil
```

Description

This creates or deletes a pinpair. A pinpair consists of two un-ordered pins or ratTs on the same net. For example, pinpair u1.2:r1.2 is the same pinpair as r1.2:u1.2. If the pinpair already exists then the existing pinpair is returned. The command fails in product tiers that do not support electrical constraints or the symbol editor.

Note: You cannot create a pinpair if both pins (or ratT) do not belong to the same xnet.

If the pinpair was created in an ECset, the ECsetDerived attribute will be t and cannot delete it. You must modify the pinpair in the associated ECset.

At database save, a pinpair must be part of a match group or have a legal electrical constraint property assigned to it. Legal electrical properties are:

- PROPAGATION_DELAY
- MIN_FIRST_SWITCH
- MAX_FINAL_SETTLE
- IMPEDANCE_RULE
- TIMING_DELAY_OVERRIDE
- RELATIVE_SKEW

RELATIVE_PROPAGATION_DELAY is stored on the RPD attrbiute as a list of lists.

See axlMatchGroupCreate for more infomation.



Tip
Using dbids is faster than using names.

Arguments

<i>o_pin1/o_pin2</i>	dbid of a pin or ratT
<i>t_pin1/t_pin2</i>	A pin name (<refdes>.<pin#>); ratT names are not supported.
<i>o_pinpair</i>	Pinpair dbid.
<i>lo_pinpair</i>	List of pinpair dbids (delete mode only).

Value Returned

Returns depending upon the mode.

nil: error

t Deletion was successful.

o_pinpair dbid of added or modified differential pair.

See also `axlPinPairSeek`, `axlPinsOfNet`, `axlMatchGroupCreate`.

Examples

Example 1: Xnet having two nets; NET1 and NET1A. This demonstrates that pinpairs are stored on the xnet.

Create pinpair using name:

```
pp = axlPinPair( "U2.13" "R1.2" )
```

Create pinpair with ratT of net NET1A:

```
ratTs = axlPinsOfNet( "NET1A" 'ratT)
pp = axlPinPair( "U2.13" car(ratTs) )
```

Allegro PCB and Package User Guide: SKILL Reference

Logic Access Functions

Verify pinpairs are both on NET1A:

```
n = car(axlSelectByName( "NET"  "NET1A" ) )
n->pinpair RETURNS nil
```

Examine the xnet (NET1):

```
xn = car(axlSelectByName( "XNET"  "NET1" ) )
xn->pinpair RETURNS (dbid:21697512 dbid:21697232)
```

Delete all pinpairs of Example 1:

```
axlPinPair(xn->pinpair)
```

axlPinPairSeek

```
axlPinPairSeek(  
    o_pin1  
    o_pin2  
) ==> o_pinpair/nil
```

Description

Given two pins or ratTs reports if they are part of a pinpair.

Arguments

<i>o_pin1/o_pin2</i>	dbid of a pin or ratT.
<i>t_pin1/t_pin2</i>	A pin name (<refdes>. <pin#>); ratT names not supported.

Value Returned

<i>o_pinpair</i>	Pinpair dbid.
nil	Pinpair for given pins does not exist.

See also `axlPinPair`.

Example

See if a pinpair for these two pins exists:

```
pp = axlPinPair( "U2.13" "R1.2" )
```

axlPinsOfNet

```
axlPinsOfNet(  
    o_net/t_net  
    g_mode  
) -> lo_pins/nil
```

Description

Returns list of pins and ratTs on a net or xnet. First argument can be either a net, xnet dbid or a net name (xnet names are not supported). Second option, *g_mode*, can be '*pin*' to return only the pins, '*ratT*' to return list of ratTs or nil to return both pins and ratTs. There is no meaning conveyed in the list of items returned.

Arguments

<i>o_net</i>	dbid of a net or xnet.
<i>t_net</i>	Name of a net (does not support xnet names).
<i>g_mode</i>	nil return both pins and ratTs of a net.
' <i>pin</i>	Return only pins.
' <i>ratT</i>	Return only the ratT's.

Value Returned

<i>lo_pins</i>	List of pins and/or ratTs on net or xnet.
nil	Nothing meeting criteria (or error, dbid not net or xnet).

Examples

All pins on GND:

```
net = car(axlSelectByName( "NET" "GND" ))  
lpins = axlPinsOfNet(net, 'pins)
```

All pins and ratTs on first xnet in design root (could be a net):

```
xnet = car( axlDBGetDesign()->xnet )  
lpins = axlPinsOfNet(xnet, nil)
```

axlRemoveNet

```
axlRemoveNet(  
    t_name  
)  
⇒ t/nil  
  
axlRemoveNet(  
    o_dbid  
)  
⇒ t/nil
```

Description

Removes a net. May either give a string with the net name to be renamed or *dbid* of an object on that net.



The net name may be used in properties. This function does not update these values.

Arguments

t_name Net name.

o_dbid *dbid* of a net.

Value Returned

t Net successfully removed.

nil No net is removed.

axlRenameNet

```
axlRenameNet(  
    t_old_name  
    t_new_name  
)  
⇒ t/nil  
  
axlRenameNet(  
    o_dbid  
    t_new_name  
)  
⇒ t/nil
```

Description

Renames a net. For the old object, may either give a string with the net name to be renamed, or *dbid* of an object on that net. Fails if the new net name already exists in the database.

```
dbid = axlSingleSelectName( "NET"  ' ("NET" ))
```

Note: This function does not refresh any *axl dbids* to reflect the new net name.



The net name may be used in properties. This function does not update these values.

Arguments

<i>t_old_name</i>	Existing net name.
<i>o_dbid</i>	<i>dbid</i> of an object on a net.
<i>t_new_name</i>	New net name (should not exist in the database.)

Value Returned

<i>t</i>	Net successfully renamed.
<i>nil</i>	Net name already exists in the database.

Example

```
axlRenameNet( "GND"  "NEWGND" )

; first verify the new net name doesn't exist
axlSetFindFilter(?enabled '("noall"  "nets"))
if(axlSingleSelectName( "NET"  '("NEWGND") )  then
    axlRenameNet(dbid  "NEWGND")
```

axlRenameRefdes

```
axlRenameRefdes(  
    t_old_name/o_oldCompDbid  
    t_new_name/o_newCompDbid  
)  
⇒ t/nil
```

Description

Renames a refdes. For either argument, may use a refdes name or a component instance.

If both refdes exist, a swap is done.

Arguments

<i>t_oldName</i>	Existing refdes name.
<i>o_oldCompDbid</i>	Component <i>dbid</i> .
<i>t_newName</i>	New refdes name.
<i>o_newCompDbid</i>	Component <i>dbid</i> .

Value Returned

<i>t</i>	Refdes successfully renamed or swapped.
<i>nil</i>	No refdes renamed or swapped due to incorrect arguments.

Example 1

```
axlRenameRefdes( "U1"  "X1" )
```

Changes refdes by name.

Example 2

```
axlSetFindFilter(?enabled '("noall"  "components") ?onButtons '(all))  
axlSingleSelectName( "COMPONENT"  "U1" )  
firstComp = car(axlGetSelSet())  
axlSingleSelectName( "COMPONENT"  "U2" )  
secondComp = car(axlGetSelSet())  
axlRenameRefdes(firstComp secondComp)
```

Swaps with starting point of two component *dbids*.

axlWriteDeviceFile

```
axlWriteDeviceFile(  
    o_compDefDbid  
    [t_output_dir]  
)  
⇒ t/nil
```

Description

Given a component definition, writes out a third party device file. Writes to the directory specified, or if no directory or `nil` is given, writes to the current directory.

Name of the file is `compDef->deviceType` in lower case with a `.txt` file extension. For example, if component definition (`compDef`) device type is `CAP1`, then the device file name is `cap1.txt`.

Also creates a `devices.map` file which is empty unless the device name has characters that are not legal as a filename.

See `netin` documentation for device file syntax.

Note: Do not use this function if you use Cadence Front-End Schematic packages.



This function overwrites existing <device> and devices.map files in the directory.

Arguments

<code>o_compDefDbid</code>	Component definition of the device file to write out.
<code>t_output_dir</code>	Directory in which to write the files. If not provided, the current directory is used.

Value Returned

<code>t</code>	Device file successfully written.
<code>nil</code>	Failed to write device file due to incorrect <code>dbid</code> or directory.

Example

```
axlWriteDeviceFile( car(axlDBGetDesign()->components)->compdef )
```

Writes device file of the definition for the first component instance off the design root.

axlWritePackageFile

```
axlWritePackageFile(  
    o_symDefDbid  
    [t_output_dir]  
)  
⇒ t/nil
```

Description

Given a symbol definition, writes out symbol .dra, .psm and associated padstack files. Works like `dump_libraries` on a single symbol definition.

The file name root is the symbol definition name. For example, if the symbol definition (symDef) name is CAPCK05, the output files created are named `capck05.psm` and `capck05.dra`, plus any padstacks (in lower case) that are part of the symbol.



This function overwrites existing files in the target directory.

Arguments

<code>o_symDefDbid</code>	Symbol definition to store on disk.
<code>t_output_dir</code>	Directory in which to store the files. If not provided, the current directory is used.

Value Returned

<code>t</code>	Files successfully written.
<code>nil</code>	Failed to write files due to incorrect <code>dbid</code> or directory.

Example

```
symDef = car(axlDBGetDesign()->components)->symbol->definition  
axlWritePackageFile( symDef )
```

Writes symbol files of the definition for the first component instance off the design root.

Building Contexts in Allegro

Introduction

A context via can be created by either of two methods – standard and autoload. Both methods substantially improve performance of Skill code loading. Even more benefits can accrue if you combine several Skill files into one context. The autoload method is a super-set of standard contexts and offers deferred context loading functionality. The autoload method is used by all Allegro provided contexts.

The standard contexts are much easier to build, are more evident to the user, and typically require more memory. The autoload contexts are much harder to build, but the system only loads the contexts upon demand. For a more complete discussion of the differences, see the section on Contexts in the *Skill Language User Guide*.

Requirements

You must have a Skill developers license and the `il_allegro` program. Currently, this license is only available on UNIX. The `il_allegro` program is part of every standard Allegro release.

Cautions

Most Skill code can be built into contexts. However, there are several potential problems that you should keep in mind when writing code. A complete discussion of these issues can be found in Chapter 10 of the *Skill Language User Guide*.

Note: Cadence recommends that you prefix your Skill functions with upper case prefixes. This minimizes the chance of naming collisions with Cadence Skill functions that use lower case prefixes.

Additionally, autoload contexts have some additional cautions. Please adhere to the following guidelines:

Autoload Context Guidelines

- Files put into an autoload context should only contain variables and procedures (functions).
- Do not load other skill files. Have `startup.il` load them.
- Do not call `axlCmdRegister`.
- Do not do anything outside of a procedure – it will not work.

Building Standard Contexts

To build a standard context

1. Create a directory that has all the Skill files to be built into the context.
2. Add the `startup.il` file (see [File B1](#) on page 932).
3. Create a Skill function with the same name as your context that registers your commands with the Allegro shell. This step is required in `allegro_designer` if you wish to access your Skill code. Only one of these functions is permitted per context. The function name must be the same as the context name. This step is analogous to the `.ini` file in autoload contexts.

Format:

```
(defun <ContextName> ()  
  (axlCmdRegister "mycommand" '<myskillcommand> ?cmdType ....)  
  .... other axlCmdRegister ..  
)
```

Example:

```
(defun MYTEST()  
  (axlCmdRegister "mytest" 'MYTest ?cmdType "general")  
)
```

4. Run the `buildcxt <ContextName>` script (see [File S1](#) on page 933). This produces a single file named, `<ContextName>.cxt`. For example: `buildcxt MYTEST`.

To load the context into `allegro_designer`, issue the Allegro command `loadcontext <ContextName>`. In programs where the Skill type-in mode is available, the Skill functions `loadContext <contextName.cxt>` and `callInitProc <ContextName>` perform the same function.

Example:

```
Allegro > loadcontext MYTEST
```

Skill version:

```
skill > (loadContext "MYTEST.txt")
skill > (callInitProc "MYTEST")
```

Building Autoload Contexts

To build a context by the autoload method

1. Create directory hierarchies:

```
./pvt/etc/context
./etc/context
```

2. Under ./pvt/etc/context, create a directory using your context name and populate it with your Skill files.
3. Add a `startup.il` (see [File B1](#) on page 932) to the mix and stir well.
4. Insure that the `ctxtFuncs.il` (see [File A1](#) on page 934) is in the root directory.
5. Run the `buildautocxt` (see [File A2](#) on page 937) UNIX command with your context name. For example: `buildautocxt <myContext>`.
6. If the context build is successful, you will have 3 files in the `./etc/context` directory with your context name (`.aux`, `.cxt`, `.toc`).
7. Add an optional fourth file with a `<myContext>.ini` that has your `axlCmdRegister`. If you do not wish to register your Skill commands as Allegro commands, you may skip this step. However, in `allegro_designer` this is the only method for accessing your Skill code.

Example:

```
(axlCmdRegister "my_command" 'MYSkillFunction ?cmdType "interactive")
```

8. Take the four context files and add them to the directory `<cds_root>/share pcb/etc/context`.
9. Edit the `pcd` file in this directory for the product requiring the context.
Names are:

allegro.pcd	All allegro_layout (CBD) based products.
designer.pcd	allegro_designer
apd.pcd	advanced_package_designer
floorplan.pcd	allegro_si

You need to add a line at the end of the file in the following format:

<NAME><VERSION><CONTEXTS>

Note: Neither the NAME of VERSION is important. It is only used with the Skill function printBlend.

Example:

MYCONTEXT 1.0MyContext

Two environment Bourne variables help in debugging problems in this area. They are:

CDS_DEBUG_CONTEXTS	file /tmp/context.log - context stats
CDS_DEBUG_CXTINIT	file /tmp/initCxt.log - context init
	- also stderr init context print

Files with This Package

File B1

Helper Skill code to load all Skill files in a directory.

```
;-----
;startup.il
foreach(file rexMatchList(".*\\.il$" getDirFiles("."))

    ; don't load myself -- bad idea
    when( nequal(file, "startup.il")
        load(file)
    )
)
```

File S1

buildcxt csh script to for building standard contexts.

```
#!/bin/csh -f
# This builds a standard context see README.cxt for other set-up requirements

if ($#argv != 1) then
    echo "Usage: $0 <context name>"
    echo "Assumes that a startup.il file exists in current directory"
    echo " this file is used to specify the loading of other skill files"
    exit 1
endif
set theContext = $argv[1]

if (!(-e startup.il)) then
    echo "ERROR: Can't find standard.il file"
    exit 1
endif

il_allegro << EOF
(setSkillPath ".")
(setContext "$theContext")
(load "startup.il")
(defInitProc "$theContext" '$${theContext}')
(saveContext "$theContext.cxt")
(exit)
EOF
/bin/rm -f AUTOSAVE.brd

echo ""
echo ""
echo ""
echo "Context will be found $theContext.cxt"
echo ""
exit 0
```

File A1

cxtFuncs.il Skill helper program to build autoload contexts.

```
; (
;-----
; EXPORTED FUNCTIONS:
;   buildContext : used to build a context
;   getContext    : used to load a context
;
;   Mods -- fxf 8/25/95 to support local building of contexts
;-----
;
; Constants
;   ilcDftSourceFileDir : directory name where Skill source
;                         files reside
;   ilcDftDeliveryDir   : directory name where delivered
;                         context files are saved.
;   (fxf) may be overridden before calling
;-----

unless(boundp('ilcDftSourceFileDir) ilcDftSourceFileDir = "pvt/etc/context")
unless(boundp('ilcDftDeliveryDir) ilcDftDeliveryDir = "etc/context")

(defun _parsePath (path)
(let (lpath)
  (cond (path
          lpath = parseString(path "/"))
        (while (!rindex(car(lpath) "tools")) lpath = cdr(lpath))
        buildString(lpath "/"))
        )
  (t nil)))
))

_stacktrace = 10
setSkillPath(strcat(". ~ " prependInstallPath("local")))
(cond ((getd 'dbSetPath) (dbSetPath ". ~")))
;

; loadCxt --
```

Allegro PCB and Package User Guide: SKILL Reference

Building Contexts in Allegro

```
; Load a context and call its init function.  
;  
(defun loadCxt (cxt ctxtPath)  
  (let ((f (strcat (cdsGetInstPath ctxtPath) "/" ctxt ".cxt"))))  
    (cond  
      ((null (isFile f)) nil)  
      ((null (loadContext f))  
       (printf "load of context %s failed\n" ctxt))  
      ((null (callInitProc ctxt))  
       (printf "init proc of context %s failed\n" ctxt))  
      (t (printf "Loading context %s\n" ctxt))  
    ))  
  )  
)  
  
;  
; buildContext --  
; Build a new context, even if one exists.  
;  
  
(defun buildContext (ctxt @rest targs)  
  (let (ctxtPath srcPath fullCxtPath)  
  
    ctxtPath = ilcDftDeliveryDir  
    (setq srcPath (strcat ilcDftSourceFileDir "/" ctxt))  
  
    ;; <fxf>: doesn't allow local contexts so use above 2 lines  
    ;;(cond ((car targs) (setq ctxtPath (car targs)))  
    ;;((setq ctxtPath (_parsePath (_iliGetActualCxtPath ctxt))) t)  
    ;;(t (setq ctxtPath ilcDftDeliveryDir))  
    ;;(cond ((cadr targs) (setq srcPath (cadr targs)))  
    ;;((setq srcPath (_parsePath (_iliGetActualSrcPath ctxt))) t)  
    ;;(t (setq srcPath (strcat ilcDftSourceFileDir "/" ctxt))))  
    fullCxtPath = cdsGetInstPath(ctxtPath)  
  
    (deleteFile (strcat fullCxtPath "/" ctxt ".cxt"))  
    (deleteFile (strcat fullCxtPath "/" ctxt ".al"))  
    (deleteFile (strcat fullCxtPath "/" ctxt ".ini"))  
  
    (updateContext ctxt ctxtPath srcPath)  
    (updateAutoloads ctxt ctxtPath srcPath)
```

Allegro PCB and Package User Guide: SKILL Reference

Building Contexts in Allegro

```
) )

;

; updateContext --
;   If there is source and it is newer than the context,
;   then build a new context. Otherwise if there is no source
;   use the existing context.
;

(defun updateContext (cxt ctxtPath srcPath)
  (cond ((isDir (cdsGetInstPath srcPath)) (makeCapContext ctxt ctxtPath srcPath))
        ((loadCxt ctxt ctxtPath) t)
        (t (printf "Can't find context %s\n" ctxt)))
)

(defun updateAutoloads (ctxt ctxtPath srcPath)
  (let ((afile (sprintf nil "%s/%s.al" (cdsGetInstPath srcPath) ctxt))
        (ifile (sprintf nil "%s/%s.ini" (cdsGetInstPath srcPath) ctxt)))

    (cond ((isFile ifile) (system (sprintf nil "cp %s %s" ifile (cdsGetInstPath
ctxtPath))))
          ((isFile afile) (system (sprintf nil "cp %s %s" afile (cdsGetInstPath
ctxtPath))))
          (t t)))
)

;

; getContext --
;   Load the context if it exists, otherwise build it.
;

(defun getContext (ctxt &rest targs)
  (let (ctxtPath srcPath)
    (cond ((car targs) (setq ctxtPath (car targs)))
          ((setq ctxtPath (_parsePath (_iliGetActualCxtPath ctxt))) t)
          (t (setq ctxtPath ilcDftDeliveryDir)))
    (cond ((cadr targs) (setq srcPath (cadr targs)))
          ((setq srcPath (_parsePath (_iliGetActualSrcPath ctxt))) t)
          (t (setq srcPath (strcat ilcDftSourceFileDir "/" ctxt)))))

    (cond ((loadCxt ctxt ctxtPath) t)
          ((isDir ctxt (cdsGetInstPath srcPath))))
```

```
(makeCapContext ctxt ctxtPath srcPath))  
(t (printf "Can't get context %s\n" ctxt)  
))  
  
(sstatus trapDefs ilcDftDeliveryDir)  
(sstatus lazyComp nil)
```

File A2

buildautocxt csh script to build autoload contexts.

```
#!/bin/csh -f  
# This builds a context see README.cxt for other set-up requirements  
  
if ($#argv != 1) then  
    echo "Usage: $0 <context name>"  
    exit 1  
endif  
set theContext = $argv[1]  
  
if (!(-e pvt/etc/context/$argv[1])) then  
    echo "pvt/etc/context/$argv[1] does not exist"  
    exit 1  
endif  
  
if (!(-e etc/context)) then  
    mkdir -p etc/context  
endif  
  
il_allegro -ilLoadIL ctxtFuncs.il << EOF  
(getContext "skillCore")  
(setSkillPath ".")  
(cdsSetInstPath ".")  
buildContext "$theContext"  
exit  
EOF  
  
echo ""
```

Allegro PCB and Package User Guide: SKILL Reference

Building Contexts in Allegro

```
echo ""
echo ""
echo "Context files will be found at etc/context/$theContext.*"
echo ""
exit 0
```