

# **Allegro User Guide: SKILL Reference**

**Product Version 17.4-2019**

**October 2019**

**Document Last Updated: March 2022**

© 1991- 2022 Cadence Design Systems, Inc. All rights reserved.

Portions © Apache Software Foundation, Sun Microsystems, Free Software Foundation, Inc., Regents of the University of California, Massachusetts Institute of Technology, University of Florida . Used by permission. Printed in the United States of America.

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

Allegro Platform Products contain technology licensed from, and copyrighted by: Apache Software Foundation, 1901 Munsey Drive Forest Hill, MD 21050, USA © 2000-2005, Apache Software Foundation. Sun Microsystems, 4150 Network Circle, Santa Clara, CA 95054 USA © 1994-2007, Sun Microsystems, Inc. Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA © 1989, 1991, Free Software Foundation, Inc. Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, © 2001, Regents of the University of California. Daniel Stenberg, © 1996 - 2006, Daniel Stenberg. UMFPACK © 2005, Timothy A. Davis, University of Florida, (davis@cise.ulv.edu). Ken Martin, Will Schroeder, Bill Lorensen © 1993-2002, Ken Martin, Will Schroeder, Bill Lorensen. Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts, USA © 2003, the Board of Trustees of Massachusetts Institute of Technology. All rights reserved.

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

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

All other trademarks are the property of their respective holders.

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

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

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

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

# **Contents**

---

<u>Alphabetical List of Functions</u> .....	35
---	----

<u>Before You Start</u> .....	57
-------------------------------	----

<u>About This Manual</u> .....	57
--------------------------------	----

<u>Prerequisites</u> .....	58
----------------------------	----

<u>Command Syntax Conventions</u> .....	58
---	----

<u>Referencing Objects by Name</u> .....	59
--	----

<u>Finding Information in This Manual</u> .....	60
---	----

<u>Other Sources of Information</u> .....	66
---	----

## **1**

<u>Introduction to Allegro PCB Editor SKILL Functions</u> .....	69
---	----

<u>Overview</u> .....	69
-----------------------	----

<u>AXL-SKILL in Allegro PCB Editor</u> .....	69
--	----

<u>AXL-SKILL Database</u> .....	72
---------------------------------	----

## **2**

<u>The Allegro PCB Editor Database User Model</u> .....	81
---	----

<u>Overview</u> .....	81
-----------------------	----

<u>Description of Database Objects</u> .....	82
--	----

<u>Figure Database Types</u> .....	87
------------------------------------	----

<u>Logical Database Types</u> .....	110
-------------------------------------	-----

<u>Property Dictionary Database Types</u> .....	118
---	-----

<u>Parameter Database Types</u> .....	121
---------------------------------------	-----

## **3**

<u>Parameter Management Functions</u> .....	131
---	-----

<u>axlcreate – Obsolete</u> .....	132
-----------------------------------	-----

<u>axIDBGetTextBlockCount</u> .....	133
-------------------------------------	-----

## Allegro SKILL Reference

---

<u><a href="#">axIDBGridGet</a></u> . . . . .	134
<u><a href="#">axIDBGridSet</a></u> . . . . .	136
<u><a href="#">axIDBTextBlockCreate</a></u> . . . . .	138
<u><a href="#">axIDBTextBlockFindName</a></u> . . . . .	140
<u><a href="#">axIDBTextBlockGetName</a></u> . . . . .	141
<u><a href="#">axIDBTextBlockSetName</a></u> . . . . .	142
<u><a href="#">axIExportXmlIDBRecords</a></u> . . . . .	143
<u><a href="#">axIFilmCreate</a></u> . . . . .	144
<u><a href="#">axImportXmlIDBRecords</a></u> . . . . .	147
<u><a href="#">axIMiniStatusReset</a></u> . . . . .	149
<u><a href="#">axIPadSuppressGet</a></u> . . . . .	150
<u><a href="#">axIPadSuppressOkLayer</a></u> . . . . .	152
<u><a href="#">axIPadSuppressSet</a></u> . . . . .	153
<u><a href="#">axIParamFilletDoc</a></u> . . . . .	156
<u><a href="#">axIParamShapeDoc</a></u> . . . . .	160
<u><a href="#">axIGetParam</a></u> . . . . .	168
<u><a href="#">axIParamDesignDoc</a></u> . . . . .	173
<u><a href="#">axISetParam</a></u> . . . . .	175
<u><a href="#">axIsParamType</a></u> . . . . .	177
<u><a href="#">axItdfIsEnabled</a></u> . . . . .	178
<u><a href="#">axItdfLoad</a></u> . . . . .	179
<u><a href="#">axItdfSave</a></u> . . . . .	180
<u><a href="#">Color Access</a></u> . . . . .	181
<u><a href="#">axIColorDoc</a></u> . . . . .	181
<u><a href="#">axIColorGet</a></u> . . . . .	184
<u><a href="#">axIColorShadowGet</a></u> . . . . .	186
<u><a href="#">axIColorShadowSet</a></u> . . . . .	188
<u><a href="#">axIColorLoad</a></u> . . . . .	190
<u><a href="#">axIColorOnGet – Obsolete</a></u> . . . . .	192
<u><a href="#">axIColorOnSet – Obsolete</a></u> . . . . .	193
<u><a href="#">axIColorPriorityGet – Obsolete</a></u> . . . . .	194
<u><a href="#">axIColorPrioritySet – Obsolete</a></u> . . . . .	195
<u><a href="#">axIColorSave</a></u> . . . . .	196
<u><a href="#">axIColorSet</a></u> . . . . .	197
<u><a href="#">axICVFCColorChooserDlg</a></u> . . . . .	200
<u><a href="#">axIClearObjectCustomColor</a></u> . . . . .	201

## Allegro SKILL Reference

---

<u><a href="#">axICustomColorObject</a></u>	203
<u><a href="#">axILayerPriorityClearAll</a></u>	205
<u><a href="#">axILayerPriorityGet</a></u>	206
<u><a href="#">axILayerPriorityRestoreAll</a></u>	208
<u><a href="#">axILayerPrioritySaveAll</a></u>	209
<u><a href="#">axILayerPrioritySet</a></u>	210
<u><a href="#">axIIsCustomColored</a></u>	212
<u><a href="#">Database Layer Management</a></u>	213
<u><a href="#">axIClasses</a></u>	213
<u><a href="#">axICompareLayers</a></u>	214
<u><a href="#">axIDBGetLayerType</a></u>	216
<u><a href="#">axIGetExternalLayers</a></u>	217
<u><a href="#">axIGetXSection – Obsolete</a></u>	218
<u><a href="#">axIIsEtchLayer</a></u>	220
<u><a href="#">axIIsLayer</a></u>	221
<u><a href="#">axIIsVisibleLayer</a></u>	222
<u><a href="#">axILayerCreateCrossSection – Obsolete</a></u>	223
<u><a href="#">axILayerCreateNonConductor</a></u>	224
<u><a href="#">axILayerDelete</a></u>	225
<u><a href="#">axILayerExternal</a></u>	227
<u><a href="#">axILayerGet</a></u>	228
<u><a href="#">axILayerRenameNonConductor</a></u>	230
<u><a href="#">axILayerViaLabel</a></u>	231
<u><a href="#">axIMaterialGet</a></u>	232
<u><a href="#">axIVisibleDesign</a></u>	235
<u><a href="#">axIVisibleGet</a></u>	237
<u><a href="#">axIVisibleLayer</a></u>	239
<u><a href="#">axIVisibleSet</a></u>	240
<u><a href="#">axIConductorBottomLayer</a></u>	241
<u><a href="#">axIConductorTopLayer</a></u>	242
<u><a href="#">axIDBCreateFilmRec – Obsolete</a></u>	243
<u><a href="#">axISetPlaneType</a></u>	244
<u><a href="#">axISubclasses</a></u>	245
<u><a href="#">axISubclassRoute</a></u>	247
<u><a href="#">axIXSectionAssign</a></u>	249
<u><a href="#">axIXSectionCopy</a></u>	250

<u><a href="#">axIXSectionCreate</a></u> .....	251
<u><a href="#">axIXSectionCreateStackup</a></u> .....	256
<u><a href="#">axIXSectionDelete</a></u> .....	257
<u><a href="#">axIXSectionDeleteStackup</a></u> .....	258
<u><a href="#">axIXSectionGet</a></u> .....	259
<u><a href="#">axIXSectionLayerFunctions</a></u> .....	266
<u><a href="#">axIXSectionLayerTypes</a></u> .....	267
<u><a href="#">axIXSectionModify</a></u> .....	268
<u><a href="#">axIXSectionRemove</a></u> .....	269
<u><a href="#">axIXSectionRename</a></u> .....	271
<u><a href="#">axIXSectionSet</a></u> .....	272

## 4

<b><u><a href="#">Selection and Find Functions</a></u></b> .....	275
<u><a href="#">Overview</a></u> .....	275
<u><a href="#">Select Set Highlighting</a></u> .....	276
<u><a href="#">Select Modes</a></u> .....	276
<u><a href="#">Finding Objects by Name</a></u> .....	276
<u><a href="#">Point Selection</a></u> .....	277
<u><a href="#">Area Selection</a></u> .....	277
<u><a href="#">Miscellaneous Select Functions</a></u> .....	278
<u><a href="#">axISelect—The General Select Function</a></u> .....	278
<u><a href="#">Select Set Management</a></u> .....	278
<u><a href="#">Find Filter Control</a></u> .....	278
<b><u><a href="#">Selection and Find Functions</a></u></b> .....	280
<u><a href="#">axISingleSelectPoint</a></u> .....	280
<u><a href="#">axIAddSelectPoint</a></u> .....	283
<u><a href="#">axISubSelectPoint</a></u> .....	285
<u><a href="#">axISingleSelectBox</a></u> .....	287
<u><a href="#">axIAddSelectBox</a></u> .....	289
<u><a href="#">axISubSelectBox</a></u> .....	290
<u><a href="#">axIAddSelectAll</a></u> .....	291
<u><a href="#">axISubSelectAll</a></u> .....	292
<u><a href="#">axISingleSelectName</a></u> .....	294
<u><a href="#">axIAddSelectName</a></u> .....	296

<u>axISubSelectName</u>	298
<u>axISingleSelectObject</u>	300
<u>axIAddSelectObject</u>	301
<u>axISubSelectObject</u>	302
<u>axISelect</u>	303
<u>axIGetSelSet</u>	306
<u>axIGetSelSetCount</u>	308
<u>axIClearSelSet</u>	309
<u>axIGetFindFilter</u>	310
<u>axISetFindFilter</u>	312
<u>axIAutoOpenFindFilter</u>	322
<u>axIOpenFindFilter</u>	323
<u>axICloseFindFilter</u>	324
<u>axIDBFindByName</u>	325
<u>axIFindFilterIsOpen</u>	327
<u>axISelectByName</u>	328
<u>axISelectByProperty</u>	333
<u>axISnapDisableAtRMB</u>	335
<u>axISnapEnableAtRMB</u>	336
<u>axISnapToObject</u>	337
<u>axILastPickIsSnapped</u>	339

## 5

<u>Interactive Edit Functions</u>	341
<u>Overview</u>	341
<u>AXL/SKILL Interactive Edit Functions</u>	342
<u>axIAllShapesMerge</u>	342
<u>axIBondFingerDelete</u>	343
<u>axIBondWireDelete</u>	344
<u>axIChangeLine2Cline</u>	345
<u>axIChangeLineFont</u>	346
<u>axIChangeWidth</u>	347
<u>axICopyProperties</u>	349
<u>axICopyObject</u>	351
<u>axIDBAltOrigin</u>	353

## Allegro SKILL Reference

---

<u><a href="#">axIDBChangeText</a></u>	.....	355
<u><a href="#">axIDBConnectItem</a></u>	.....	358
<u><a href="#">axIDeleteObject</a></u>	.....	360
<u><a href="#">axIDeleteByRectangle</a></u>	.....	363
<u><a href="#">axIDeleteTaper</a></u>	.....	365
<u><a href="#">axIDBDeleteProp</a></u>	.....	366
<u><a href="#">axIDBDeletePropAll</a></u>	.....	369
<u><a href="#">axIDBDeletePropDictEntry</a></u>	.....	370
<u><a href="#">axIDBOpenShape</a></u>	.....	371
<u><a href="#">axIDeleteFillet</a></u>	.....	373
<u><a href="#">axIFillet</a></u>	.....	374
<u><a href="#">axIFilletConvert</a></u>	.....	375
<u><a href="#">axIGetLastEnterPoint</a></u>	.....	376
<u><a href="#">axILastPick</a></u>	.....	377
<u><a href="#">axIWindowBoxGet</a></u>	.....	378
<u><a href="#">axIWindowBoxSet</a></u>	.....	379
<u><a href="#">axIReplacePadstack</a></u>	.....	380
<u><a href="#">axIPurgePadstacks</a></u>	.....	381
<u><a href="#">axIPurge3DModelMapDataInDesign</a></u>	.....	383
<u><a href="#">axIShapeAutoVoid</a></u>	.....	384
<u><a href="#">axIShapeChangeDynamicType</a></u>	.....	386
<u><a href="#">axIShapeDeleteVoids</a></u>	.....	388
<u><a href="#">axIShapeDynamicUpdate</a></u>	.....	390
<u><a href="#">axIShapeRaisePriority</a></u>	.....	391
<u><a href="#">axIShapeMerge</a></u>	.....	393
<u><a href="#">axIShovelItems</a></u>	.....	395
<u><a href="#">axIShoveSetParams</a></u>	.....	396
<u><a href="#">axISmoothDesign</a></u>	.....	399
<u><a href="#">axISmoothItems</a></u>	.....	400
<u><a href="#">axISmoothSetParams</a></u>	.....	401
<u><a href="#">axIStepDelete</a></u>	.....	404
<u><a href="#">axIStepSet</a></u>	.....	405
<u><a href="#">axISymbolAttach</a></u>	.....	407
<u><a href="#">axISymbolDetach</a></u>	.....	409
<u><a href="#">axIAddTaper</a></u>	.....	411
<u><a href="#">axITextOrientationCopy</a></u>	.....	412

<u><a href="#">axITransformObject</a></u> . . . . .	413
<u><a href="#">axIZoneDelete</a></u> . . . . .	417
<u><a href="#">axIZoneSet</a></u> . . . . .	418
<u><a href="#">Padstack Access Functions</a></u> . . . . .	420
<u><a href="#">axIDBCreatePadStack</a></u> . . . . .	420
<u><a href="#">axIPadFigureTypes</a></u> . . . . .	428
<u><a href="#">axIPadstackEdit</a></u> . . . . .	429
<u><a href="#">axIPadstackSetType</a></u> . . . . .	439
<u><a href="#">axIPadstackUsageTypes</a></u> . . . . .	441
<u><a href="#">axIPadUserMaskLayers</a></u> . . . . .	442
 <b>6</b>	
<u><a href="#">Database Read Functions</a></u> . . . . .	445
<u><a href="#">AXL-SKILL Database Read Functions</a></u> . . . . .	445
<u><a href="#">axIAltSymbolList</a></u> . . . . .	446
<u><a href="#">axIAltSymbolOK</a></u> . . . . .	447
<u><a href="#">axIAltSymbolReplace</a></u> . . . . .	448
<u><a href="#">axIBackdrillGet</a></u> . . . . .	450
<u><a href="#">axIDBGetDesign</a></u> . . . . .	453
<u><a href="#">axIDBGetDrillPlating</a></u> . . . . .	455
<u><a href="#">axIIsDBIDType</a></u> . . . . .	456
<u><a href="#">axIDBGetAttachedText</a></u> . . . . .	458
<u><a href="#">axIDBGetPad</a></u> . . . . .	460
<u><a href="#">axIDBGetPropDict</a></u> . . . . .	463
<u><a href="#">axIDBGetPropDictEntry</a></u> . . . . .	464
<u><a href="#">axIDBGetProperties</a></u> . . . . .	466
<u><a href="#">axIDBGetDesignUnits</a></u> . . . . .	468
<u><a href="#">axIDBRefreshId</a></u> . . . . .	469
<u><a href="#">axIDBGetLonelyBranches</a></u> . . . . .	471
<u><a href="#">axIDBGetConnect</a></u> . . . . .	472
<u><a href="#">axIDBIsBondpad</a></u> . . . . .	474
<u><a href="#">axIDBIsBondwire</a></u> . . . . .	475
<u><a href="#">axIDBIsFixed</a></u> . . . . .	476
<u><a href="#">axIDBIsPackagePin</a></u> . . . . .	478
<u><a href="#">axIDBViaStack</a></u> . . . . .	479

## Allegro SKILL Reference

---

<u><a href="#">axIGetModuleInstanceDefinition</a></u>	480
<u><a href="#">axIGetModuleInstanceLocation</a></u>	481
<u><a href="#">axIGetModuleInstanceLogicMethod</a></u>	482
<u><a href="#">axIGetModuleInstanceNetExceptions</a></u>	483
<u><a href="#">axIIIsDummyNet</a></u>	484
<u><a href="#">axIIIsLayerNegative</a></u>	485
<u><a href="#">axIIIsPinUnused</a></u>	486
<u><a href="#">axIIIsitFill</a></u>	487
<u><a href="#">axIOK2Void</a></u>	488
<u><a href="#">axIDBDynamicShapes</a></u>	489
<u><a href="#">axIDBGetShapes</a></u>	490
<u><a href="#">axIDBTextBlockCompact</a></u>	491
<u><a href="#">axIShapeArea</a></u>	492
<u><a href="#">axIStepGet</a></u>	493
<u><a href="#">axIStepMappedInstance</a></u>	496
<u><a href="#">axIZoneAccess</a></u>	497

## 7

<u><a href="#">Allegro PCB Editor Interface Functions</a></u>	499
<u><a href="#">Overview</a></u>	499
<u><a href="#">AXL-SKILL Interface Function Examples</a></u>	499
<u><a href="#">Allegro PCB Editor Interface Functions</a></u>	506
<u><a href="#">axIClearDynamics</a></u>	506
<u><a href="#">axIAddSimpleRbandDynamics</a></u>	507
<u><a href="#">axIAddSimpleMoveDynamics</a></u>	510
<u><a href="#">axIDesignFlip</a></u>	512
<u><a href="#">axIEEnterPoint</a></u>	513
<u><a href="#">axIEEnterString</a></u>	514
<u><a href="#">axIEEnterAngle</a></u>	515
<u><a href="#">axICancelEnterFun</a></u>	516
<u><a href="#">axIFinishEnterFun</a></u>	517
<u><a href="#">axIGetDynamicsSegs</a></u>	518
<u><a href="#">axIGetLineLock</a></u>	519
<u><a href="#">axIEEnterBox</a></u>	521
<u><a href="#">axIEEnterPath</a></u>	523

## Allegro SKILL Reference

---

<u>axlHighlightObject</u>	524
<u>axlDehighlightObject</u>	526
<u>axlMiniStatusLoad</u>	527
<u>axlMiniStatusClose</u>	530
<u>axlDrawObject</u>	531
<u>axlDynamicsObject</u>	532
<u>axlEraseObject</u>	533
<u>axlControlRaise</u>	534
<u>axlEnterEvent</u>	535
<u>axlEventSetStartPopup</u>	539
<u>axlGetTrapBox</u>	541
<u>axlRatsnestBlank</u>	542
<u>axlRatsnestDisplay</u>	543
<u>axlSetDynamicsMirror</u>	544
<u>axlSetDynamicsRotation</u>	545
<u>axlShowObjectToFile</u>	546
<u>axlUICmdPopupSet</u>	547
<u>axlWindowFit</u>	548
<u>axlZoomBbox</u>	549
<u>axlZoomCenter</u>	550
<u>axlZoomControl</u>	551
<u>axlZoomFit</u>	553
<u>axlZoomInOut</u>	554
<u>axlZoomPoints</u>	555
<u>axlZoomToDbid</u>	556
<u>axlZoomWorld</u>	558
<u>axlMakeDynamicsPath</u>	559

## 8

<u>Allegro PCB Editor Command Shell Functions</u>	563
<u>Command Shell Functions</u>	563
<u>axlGetAlias</u>	564
<u>axlGetFunckey</u>	565
<u>axlGetVariable</u>	566
<u>axlGetVariableList</u>	568

<u>axlJournal</u>	570
<u>axlProtectAlias</u>	572
<u>axlIsProtectAlias</u>	573
<u>axlReadOnlyVariable</u>	574
<u>axlSetAlias</u>	576
<u>axlSetAlias</u>	578
<u>axlSetFuncKey</u>	580
<u>axlSetVariable</u>	582
<u>axlSetVariableFile</u>	584
<u>axlShell</u>	585
<u>axlShellPost</u>	586
<u>axlUnsetVariable</u>	588
<u>axlUnsetVariableFile</u>	589

## 9

<u>IC Packaging Commands</u>	591
<u>axlChangeLayer</u>	592
<u>axlCNSAssemblyModeGet</u>	594
<u>axlCNSAssemblyModeSet</u>	596
<u>axlCNSGetAssembly</u>	598
<u>axlCNSSetAssembly</u>	601
<u>axlCreateDeviceFileTemplate</u>	604
<u>axlCompAddPin</u>	605
<u>axlCompDeletePin</u>	608
<u>axlCompMovePin</u>	609
<u>axlComponentChangeClass</u>	611
<u>axlCompSetPinAttributes</u>	613
<u>axlDBIsBondingWireLayer – Obsolete</u>	615
<u>axlDBIsDiePad</u>	616
<u>axlDBIsPlatingbarPin</u>	617
<u>axlGetDieType</u>	618
<u>axlGetMetalUsageForLayer</u>	619
<u>axlGetWireProfileDefinition</u>	621
<u>axlAddAutoAssignNetAlgorithm</u>	622
<u>axlGetWireProfileDirection</u>	623

## Allegro SKILL Reference

---

<u><a href="#">axlGetAllVisibleProfiles</a></u>	.....	624
<u><a href="#">axlSetAllProfilesVisible</a></u>	.....	625
<u><a href="#">axlGetBondWireLength</a></u>	.....	626
<u><a href="#">axlLoadViaStructure</a></u>	.....	627
<u><a href="#">axlSetBondWireProfile</a></u>	.....	628
<u><a href="#">axlImportWireProfileDefinitions</a></u>	.....	629
<u><a href="#">axlSetBondWireProfile</a></u>	.....	630
<u><a href="#">axlSetDieStackData</a></u>	.....	631
<u><a href="#">axlDBIsDieStackLayer</a></u>	.....	632
<u><a href="#">axlGetDieData</a></u>	.....	633
<u><a href="#">axlGetDieStackData</a></u>	.....	636
<u><a href="#">axlGetDieStackMemberSet</a></u>	.....	638
<u><a href="#">axlGetDieStackNames</a></u>	.....	640
<u><a href="#">axlGetIposerData</a></u>	.....	641
<u><a href="#">axlGetSpacerData</a></u>	.....	643
<u><a href="#">axlGetWireProfileColor</a></u>	.....	645
<u><a href="#">axlGetWireProfileVisible</a></u>	.....	646
<u><a href="#">axlPackageDesignCheckAddCategory</a></u>	.....	647
<u><a href="#">axlPackageDesignCheckAddCheck</a></u>	.....	649
<u><a href="#">axlPackageDesignCheckDrcError</a></u>	.....	652
<u><a href="#">axlPackageDesignCheckLogError</a></u>	.....	653
<u><a href="#">axlSetDieData</a></u>	.....	655
<u><a href="#">axlSetDieType</a></u>	.....	656
<u><a href="#">axlSetIposerData</a></u>	.....	657
<u><a href="#">axlSetSpacerData</a></u>	.....	658
<u><a href="#">axlSetWireProfileColor</a></u>	.....	659
<u><a href="#">axlSetWireProfileVisible</a></u>	.....	660
<u><a href="#">axlCDLNetlistSetValue</a></u>	.....	661
 <u><a href="#">10</a></u>		
<u><a href="#">User Interface Functions</a></u>	.....	663
<u><a href="#">Window Placement</a></u>	.....	663
<u><a href="#">Using Menu Files</a></u>	.....	664
<u><a href="#">Dynamically Loading Menus</a></u>	.....	666
<u><a href="#">Understanding the Menu File Format</a></u>	.....	667

## Allegro SKILL Reference

---

<u><a href="#">AXL-SKILL User Interface Functions</a></u> . . . . .	673
<u><a href="#">axlCancelOff</a></u> . . . . .	673
<u><a href="#">axlCancelOn</a></u> . . . . .	674
<u><a href="#">axlCancelTest</a></u> . . . . .	676
<u><a href="#">axlCancelClearFormClosable</a></u> . . . . .	677
<u><a href="#">axlCancelSetFormClosable</a></u> . . . . .	678
<u><a href="#">axlClipboardGetText</a></u> . . . . .	680
<u><a href="#">axlClipboardSetText</a></u> . . . . .	681
<u><a href="#">axlCursorGet</a></u> . . . . .	682
<u><a href="#">axlCursorWarp</a></u> . . . . .	683
<u><a href="#">axlMeterCreate</a></u> . . . . .	684
<u><a href="#">axlMeterDestroy</a></u> . . . . .	686
<u><a href="#">axlMeterIsCancelled</a></u> . . . . .	687
<u><a href="#">axlMeterUpdate</a></u> . . . . .	688
<u><a href="#">axlUIAppMode</a></u> . . . . .	689
<u><a href="#">axlUIMenuLoad</a></u> . . . . .	691
<u><a href="#">axlUIMenuDump</a></u> . . . . .	692
<u><a href="#">axlUIColorDialog</a></u> . . . . .	693
<u><a href="#">axlUIConfirm</a></u> . . . . .	694
<u><a href="#">axlUIConfirmEx</a></u> . . . . .	695
<u><a href="#">axlUIControl</a></u> . . . . .	697
<u><a href="#">axlUIMenuChange</a></u> . . . . .	699
<u><a href="#">axlUIMenuDebug</a></u> . . . . .	701
<u><a href="#">axlUIMenuDelete</a></u> . . . . .	702
<u><a href="#">axlUIMenuFind</a></u> . . . . .	703
<u><a href="#">axlUIMenuInsert</a></u> . . . . .	706
<u><a href="#">axlUIMenuRegister</a></u> . . . . .	709
<u><a href="#">axlUIPopupMenuDump</a></u> . . . . .	711
<u><a href="#">axlUIPrompt</a></u> . . . . .	712
<u><a href="#">axlUIWCloseAll</a></u> . . . . .	714
<u><a href="#">axlUIWIconify</a></u> . . . . .	715
<u><a href="#">axlUIWIslIconic</a></u> . . . . .	716
<u><a href="#">axlUIWIslWindow</a></u> . . . . .	717
<u><a href="#">axlUIWMove</a></u> . . . . .	718
<u><a href="#">axlUIWRedraw</a></u> . . . . .	719
<u><a href="#">axlUIWSize</a></u> . . . . .	720

## Allegro SKILL Reference

---

<u>axIIsViewFileType</u>	721
<u>axIUIViewFileCreate</u>	722
<u>axIUIViewFileReuse</u>	724
<u>axIUIYesNo</u>	726
<u>axIUIWExpose</u>	728
<u>axIUIWClose</u>	729
<u>axIUIWHelpRegister</u>	731
<u>axIUIWPrint</u>	733
<u>axIUIWRedraw</u>	734
<u>axIUIWBlock</u>	735
<u>axIUIEditFile</u>	736
<u>axIUIMultipleChoice</u>	738
<u>axIUIViewFileScrollTo</u>	739
<u>axIUIWBeep</u>	740
<u>axIUIWDisableQuit</u>	741
<u>axIUIWExposeByName</u>	742
<u>axIUIWPerm</u>	743
<u>axIUIWSetHelpTag</u>	745
<u>axIUIWSetParent</u>	746
<u>axIUIWShow</u>	747
<u>axIUIWTimerAdd</u>	748
<u>axIUIWTimerRemove</u>	751
<u>axIUIWUpdate</u>	752
<u>axIUIYesNoCancel</u>	753
<u>axUIDataBrowse</u>	754

## 11

<u>Form Interface Functions</u>	757
<u>Overview</u>	757
<u>Programming</u>	757
<u>Field / Control</u>	758
<u>Using Forms Specification Language</u>	767
<u>Moving and Sizing Form Controls During Form Resizing</u>	776
<u>Using Grids</u>	781
<u>AXL-SKILL Form Interface Functions</u>	798

## Allegro SKILL Reference

---

<u><a href="#">axlFormBNFDoc</a></u>	.....	798
<u><a href="#">axlFormCallback</a></u>	.....	814
<u><a href="#">axlFormCreate</a></u>	.....	819
<u><a href="#">axlFormClearMouseActive</a></u>	.....	822
<u><a href="#">axlFormClose</a></u>	.....	823
<u><a href="#">axlFormDisplay</a></u>	.....	824
<u><a href="#">axlFormBuildPopup</a></u>	.....	825
<u><a href="#">axlFormGetField</a></u>	.....	829
<u><a href="#">axlFormGetFieldRange</a></u>	.....	831
<u><a href="#">axlFormGetOptionValue</a></u>	.....	832
<u><a href="#">axlFormGridGetSize</a></u>	.....	833
<u><a href="#">axlFormListGetCount</a></u>	.....	834
<u><a href="#">axlFormGridSelected</a></u>	.....	835
<u><a href="#">axlFormGridSelectedCnt</a></u>	.....	836
<u><a href="#">axlFormGridSetSelectRows</a></u>	.....	837
<u><a href="#">axlFormListDeleteAll</a></u>	.....	839
<u><a href="#">axlFormListSelect</a></u>	.....	842
<u><a href="#">axlFormSetEventAction</a></u>	.....	843
<u><a href="#">axlFormSetField</a></u>	.....	845
<u><a href="#">axlFormSetFieldHelpTip</a></u>	.....	848
<u><a href="#">axlFormSetInfo</a></u>	.....	849
<u><a href="#">axlFormSetMouseActive</a></u>	.....	850
<u><a href="#">axlFormTest</a></u>	.....	851
<u><a href="#">axlFormRestoreField</a></u>	.....	852
<u><a href="#">axlFormTitle</a></u>	.....	854
<u><a href="#">axllsFormType</a></u>	.....	855
<u><a href="#">axlFormSetFieldVisible</a></u>	.....	856
<u><a href="#">axlFormIsFieldVisible</a></u>	.....	857
<u><a href="#">Callback Procedure: formCallback</a></u>	.....	858
<u><a href="#">axlFormAutoResize</a></u>	.....	863
<u><a href="#">axlFormColorize</a></u>	.....	864
<u><a href="#">axlFormGetActiveField</a></u>	.....	867
<u><a href="#">axlFormGridBatch</a></u>	.....	868
<u><a href="#">axlFormGridCancelPopup</a></u>	.....	869
<u><a href="#">axlFormGridDeleteRows</a></u>	.....	870
<u><a href="#">axlFormGridEvents</a></u>	.....	871

## Allegro SKILL Reference

---

<a href="#"><u>axlFormGridGetCell</u></a>	874
<a href="#"><u>axlFormGridInsertCol</u></a>	876
<a href="#"><u>axlIsGridCellType</u></a>	879
<a href="#"><u>axlFormGridInsertRows</u></a>	880
<a href="#"><u>axlFormGridNewCell</u></a>	881
<a href="#"><u>axlFormGridReset</u></a>	882
<a href="#"><u>axlFormGridSetBatch</u></a>	883
<a href="#"><u>axlFormGridUpdate</u></a>	886
<a href="#"><u>axlFormInvalidateField</u></a>	887
<a href="#"><u>axlFormIsFieldEditable</u></a>	888
<a href="#"><u>axlFormListAddItem</u></a>	889
<a href="#"><u>axlFormListDeleteItem</u></a>	891
<a href="#"><u>axlFormListGetItem</u></a>	893
<a href="#"><u>axlFormListGetSelCount</u></a>	894
<a href="#"><u>axlFormListGetSelItems</u></a>	895
<a href="#"><u>axlFormListOptions</u></a>	896
<a href="#"><u>axlFormListSelAll</u></a>	898
<a href="#"><u>axlFormMsg</u></a>	899
<a href="#"><u>axlFormGetFieldRange</u></a>	901
<a href="#"><u>axlFormGetFieldType</u></a>	902
<a href="#"><u>axlFormDefaultButton</u></a>	903
<a href="#"><u>axlFormGridOptions</u></a>	905
<a href="#"><u>axlFormSetActiveField</u></a>	907
<a href="#"><u>axlFormSetDecimal</u></a>	908
<a href="#"><u>axlFormSetFieldEditable</u></a>	909
<a href="#"><u>axlFormSetFieldLimits</u></a>	910
<a href="#"><u>axlFormTreeViewAddItem</u></a>	911
<a href="#"><u>axlFormTreeViewChangeImages</u></a>	914
<a href="#"><u>axlFormTreeViewChangeLabel</u></a>	916
<a href="#"><u>axlFormTreeViewGetImages</u></a>	917
<a href="#"><u>axlFormTreeViewGetLabel</u></a>	918
<a href="#"><u>axlFormTreeViewGetParents</u></a>	919
<a href="#"><u>axlFormTreeViewGetSelectState</u></a>	920
<a href="#"><u>axlFormTreeViewLoadBitmaps</u></a>	921
<a href="#"><u>axlFormTreeViewSet</u></a>	923
<a href="#"><u>axlFormTreeViewSetSelectState</u></a>	926

### 12

<u>Simple Graphics Drawing Functions</u> .....	929
<u>Functions</u> .....	933
<a href="#">axIGRPDrwBitmap</a> .....	933
<a href="#">axIGRPDrwCircle</a> .....	934
<a href="#">axIGRPDrwInit</a> .....	935
<a href="#">axIGRPDrwLine</a> .....	936
<a href="#">axIGRPDrwMapWindow</a> .....	937
<a href="#">axIGRPDrwPoly</a> .....	938
<a href="#">axIGRPDrwRectangle</a> .....	939
<a href="#">axIGRPDrwText</a> .....	940
<a href="#">axIGRPDrwUpdate</a> .....	941

### 13

<u>Message Handler Functions</u> .....	943
<u>Overview</u> .....	943
<u>Message Handler Functions</u> .....	946
<a href="#">axIMsgPut</a> .....	946
<a href="#">axIMsgContextPrint</a> .....	947
<a href="#">axIMsgContextGetString</a> .....	948
<a href="#">axIMsgContextGet</a> .....	949
<a href="#">axIMsgContextTest</a> .....	950
<a href="#">axIMsgContextInBuf</a> .....	951
<a href="#">axIMsgContextRemove</a> .....	952
<a href="#">axIMsgContextStart</a> .....	953
<a href="#">axIMsgContextFinish</a> .....	954
<a href="#">axIMsgContextClear</a> .....	955
<a href="#">axIMsgCancelPrint</a> .....	956
<a href="#">axIMsgCancelSeen</a> .....	957
<a href="#">axIMsgClear</a> .....	958
<a href="#">axIMsgSet</a> .....	959
<a href="#">axIMsgTest</a> .....	960

### 14

<u>Design Control Functions</u> .....	961
<u>AXL-SKILL Design Control Functions</u> .....	961
<u>axlCurrentDesign</u> .....	962
<u>axlDesignType</u> .....	963
<u>axlCompileSymbol</u> .....	965
<u>axlSetSymbolType</u> .....	966
<u>axlDBControl</u> .....	967
<u>axlDBGetExtents</u> .....	974
<u>axlDBIgnoreFixed</u> .....	975
<u>axlDBIsReadOnly</u> .....	977
<u>axlDBSectorSize - Obsolete</u> .....	978
<u>axlGetDrawingName</u> .....	979
<u>axlIgnoreFixed</u> .....	980
<u>axlInTrigger</u> .....	981
<u>axlInTriggerFunc</u> .....	982
<u>axlIsSymbolEditor</u> .....	983
<u>axlKillDesign</u> .....	984
<u>axlOpenDesign</u> .....	985
<u>axlOpenDesignForBatch</u> .....	987
<u>axlRenameDesign</u> .....	989
<u>axlSaveDesign</u> .....	990
<u>axlSaveEnable</u> .....	992
<u>axlDBChangeDesignExtents</u> .....	993
<u>axlDBChangeDesignOrigin</u> .....	994
<u>axlDBChangeDesignUnits</u> .....	995
<u>axlDBCheck</u> .....	998
<u>axlDBCopyPadstack</u> .....	1000
<u>axlDBDelLock</u> .....	1002
<u>axlDBGetLock</u> .....	1003
<u>axlDBMemoryReclaim</u> .....	1004
<u>axlDBSetLock</u> .....	1006
<u>axlDBTuneSectorSize</u> .....	1009
<u>axlTechnologyType</u> .....	1010
<u>axlTriggerClear</u> .....	1011

<u><a href="#">axITriggerPrint</a></u> .....	1012
<u><a href="#">axITriggerSet</a></u> .....	1013
<u><a href="#">axIGetActiveLayer – Obsolete</a></u> .....	1018
<u><a href="#">axIGetActiveTextBlock – Obsolete</a></u> .....	1019
<u><a href="#">axISetActiveLayer – Obsolete</a></u> .....	1020
<u><a href="#">axIWFMAnyExported</a></u> .....	1021
<u><a href="#">axIDBDisplayControl</a></u> .....	1022
 <b>15</b>	
<b><u><a href="#">Database Create Functions</a></u></b> .....	1029
<u><a href="#">Overview</a></u> .....	1029
<u><a href="#">Path Functions</a></u> .....	1030
<u><a href="#">axIPathStart</a></u> .....	1032
<u><a href="#">axIPathArcRadius</a></u> .....	1033
<u><a href="#">axIPathArcAngle</a></u> .....	1033
<u><a href="#">axIPathArcCenter</a></u> .....	1033
<u><a href="#">axIPathLine</a></u> .....	1037
<u><a href="#">axIPathGetWidth</a></u> .....	1038
<u><a href="#">axIPathSegGetWidth</a></u> .....	1039
<u><a href="#">axIPathGetPathSegs</a></u> .....	1040
<u><a href="#">axIPathGetLastPathSeg</a></u> .....	1041
<u><a href="#">axIPathSegGetEndPoint</a></u> .....	1042
<u><a href="#">axIPathSegGetArcCenter</a></u> .....	1043
<u><a href="#">axIPathSegGetArcClockwise</a></u> .....	1044
<u><a href="#">axIPathStartCircle</a></u> .....	1045
<u><a href="#">axIPathOffset</a></u> .....	1046
<u><a href="#">axIDB2Path</a></u> .....	1047
<u><a href="#">axIDBCreatePath</a></u> .....	1048
<u><a href="#">axIDBCreateLine</a></u> .....	1051
<u><a href="#">axIDBCreateCircle</a></u> .....	1053
<u><a href="#">Create Shape Interface</a></u> .....	1054
<u><a href="#">axIDBComposeShapesFromLines</a></u> .....	1056
<u><a href="#">axIDBCreateBoundingShape</a></u> .....	1059
<u><a href="#">axIDBCreateOpenShape</a></u> .....	1062
<u><a href="#">axIDBCreateCloseShape</a></u> .....	1066

## Allegro SKILL Reference

---

<u><a href="#">axIDBActiveShape</a></u> . . . . .	1067
<u><a href="#">axIDBCreateVoidCircle</a></u> . . . . .	1068
<u><a href="#">axIDBCreateVoid</a></u> . . . . .	1069
<u><a href="#">axIDBCreateShape</a></u> . . . . .	1071
<u><a href="#">axIDBCreateRectangle</a></u> . . . . .	1073
<u><a href="#">axIDBCreateVectorizedEllipse</a></u> . . . . .	1075
<u><a href="#">Nonpath DBCreate Functions</a></u> . . . . .	1078
<u><a href="#">axICreateBondFinger</a></u> . . . . .	1078
<u><a href="#">axICreateBondWire</a></u> . . . . .	1080
<u><a href="#">axIDBCreateExternalDRC</a></u> . . . . .	1082
<u><a href="#">axIDBCreateFillet</a></u> . . . . .	1086
<u><a href="#">axIDBCreatePin</a></u> . . . . .	1088
<u><a href="#">axIDBCreateSymbol</a></u> . . . . .	1092
<u><a href="#">axIDBCreateSymbolSkeleton</a></u> . . . . .	1096
<u><a href="#">axIDBCreateText</a></u> . . . . .	1101
<u><a href="#">axIDBCreateVia</a></u> . . . . .	1104
<u><a href="#">axIDBCreateSymbolAutosilk</a></u> . . . . .	1107
<u><a href="#">axIDBCreateViaStructure</a></u> . . . . .	1108
<u><a href="#">axICreateWirebondGuide</a></u> . . . . .	1110
<u><a href="#">axIZoneCreate</a></u> . . . . .	1111
<u><a href="#">Property Functions</a></u> . . . . .	1113
<u><a href="#">axIDBCreatePropDictEntry</a></u> . . . . .	1113
<u><a href="#">axIDBAddProp</a></u> . . . . .	1117
<u><a href="#">Load and Save Functions</a></u> . . . . .	1122
<u><a href="#">axILoadPadstack</a></u> . . . . .	1122
<u><a href="#">axILoadSymbol</a></u> . . . . .	1123
<u><a href="#">axIPadstackToDisk</a></u> . . . . .	1125
<u><a href="#">axIRefreshSymbol</a></u> . . . . .	1126
<b><u>16</u></b>	
<b><u>Database Group Functions</u></b> . . . . .	1129
<u><a href="#">Overview</a></u> . . . . .	1129
<u><a href="#">axIDBAddGroupObjects</a></u> . . . . .	1130
<u><a href="#">axIDBCreateGroup</a></u> . . . . .	1131
<u><a href="#">axIDBDisbandGroup</a></u> . . . . .	1135

<u><a href="#">axIDBGetGroupFromItem</a></u> . . . . .	1136
<u><a href="#">axIDBGroupRename</a></u> . . . . .	1138
<u><a href="#">axIDBRemoveGroupObjects</a></u> . . . . .	1139
<u><a href="#">axINetClassAdd</a></u> . . . . .	1140
<u><a href="#">axINetClassCreate</a></u> . . . . .	1142
<u><a href="#">axINetClassDelete</a></u> . . . . .	1144
<u><a href="#">axINetClassGet</a></u> . . . . .	1145
<u><a href="#">axINetClassRemove</a></u> . . . . .	1147
<u><a href="#">axIRegionAdd</a></u> . . . . .	1148
<u><a href="#">axIRegionCreate</a></u> . . . . .	1150
<u><a href="#">axIRegionDelete</a></u> . . . . .	1151
<u><a href="#">axIRegionRemove</a></u> . . . . .	1152

## 17

<u><a href="#">Database Attachment Functions</a></u> . . . . .	1155
<u><a href="#">Overview</a></u> . . . . .	1155
<u><a href="#">axICreateAttachment</a></u> . . . . .	1156
<u><a href="#">axIDeleteAttachment</a></u> . . . . .	1159
<u><a href="#">axIGetAllAttachmentNames</a></u> . . . . .	1160
<u><a href="#">axIGetAttachment</a></u> . . . . .	1161
<u><a href="#">axIIsAttachment</a></u> . . . . .	1163
<u><a href="#">axISetAttachment</a></u> . . . . .	1164

## 18

<u><a href="#">Database Transaction Functions</a></u> . . . . .	1167
<u><a href="#">axIDBCloak</a></u> . . . . .	1167
<u><a href="#">axIDBTransactionCommit</a></u> . . . . .	1171
<u><a href="#">axIDBTransactionMark</a></u> . . . . .	1172
<u><a href="#">axIDBTransactionOops</a></u> . . . . .	1173
<u><a href="#">axIDBTransactionRollback</a></u> . . . . .	1174
<u><a href="#">axIDBTransactionStart</a></u> . . . . .	1175

### 19

<u>Constraint Management Functions</u> .....	1179
<u>Overview</u> .....	1179
<a href="#"><u>axICnsAddVia</u></a> .....	1180
<a href="#"><u>axICnsAssignPurge – Obsolete</u></a> .....	1181
<a href="#"><u>axICnsClassTableChange</u></a> .....	1182
<a href="#"><u>axICnsClassTableCreate</u></a> .....	1184
<a href="#"><u>axICnsClassTableDelete</u></a> .....	1186
<a href="#"><u>axICnsClassTableFind</u></a> .....	1187
<a href="#"><u>axICnsClassTableSeek</u></a> .....	1189
<a href="#"><u>axICNSCreate</u></a> .....	1191
<a href="#"><u>axICNSCsetLock</u></a> .....	1193
<a href="#"><u>axICNSDelete</u></a> .....	1195
<a href="#"><u>axICnsDeleteClassClassObjects</u></a> .....	1196
<a href="#"><u>axICnsDeleteRegionClassClassObjects</u></a> .....	1197
<a href="#"><u>axICnsDeleteRegionClassObjects</u></a> .....	1198
<a href="#"><u>axICnsDeleteVia</u></a> .....	1199
<a href="#"><u>axICNSDesignModeGet</u></a> .....	1201
<a href="#"><u>axICNSDesignModeSet</u></a> .....	1204
<a href="#"><u>axICNSDesignValueCheck</u></a> .....	1207
<a href="#"><u>axICNSDesignValueGet</u></a> .....	1208
<a href="#"><u>axICNSDesignValueSet</u></a> .....	1211
<a href="#"><u>axICNSDFAExport</u></a> .....	1213
<a href="#"><u>axICNSDFAImport</u></a> .....	1214
<a href="#"><u>axICNSDFAMode</u></a> .....	1215
<a href="#"><u>axICNSEcsetCreate</u></a> .....	1217
<a href="#"><u>axICNSEcsetDelete</u></a> .....	1219
<a href="#"><u>axICNSEcsetGet</u></a> .....	1220
<a href="#"><u>axICNSEcsetModeGet</u></a> .....	1221
<a href="#"><u>axICNSEcsetModeSet</u></a> .....	1223
<a href="#"><u>axICNSEcsetValueCheck</u></a> .....	1226
<a href="#"><u>axICNSEcsetValueGet</u></a> .....	1227
<a href="#"><u>axICNSGetDefaultMinLineWidth</u></a> .....	1230
<a href="#"><u>axICNSGetPhysical</u></a> .....	1231
<a href="#"><u>axICNSGetPinDelayEnabled</u></a> .....	1234

## Allegro SKILL Reference

---

<u><a href="#">axICNSGetPinDelayPVF</a></u>	.....	1235
<u><a href="#">axICNSGetSameNet</a></u>	.....	1236
<u><a href="#">axICNSGetSameNetXtalkEnabled</a></u>	.....	1239
<u><a href="#">axICNSGetSpacing</a></u>	.....	1240
<u><a href="#">axICNSGetViaZEnabled</a></u>	.....	1243
<u><a href="#">axICNSGetViaZPVF</a></u>	.....	1244
<u><a href="#">axICNSPhysicalModeGet</a></u>	.....	1245
<u><a href="#">axICNSIsCsetLocked</a></u>	.....	1247
<u><a href="#">axICNSIsLockedDomain</a></u>	.....	1248
<u><a href="#">axICNSLockDomain</a></u>	.....	1250
<u><a href="#">axICNSOptions</a></u>	.....	1251
<u><a href="#">axICNSPhysicalModeSet</a></u>	.....	1253
<u><a href="#">axICNSSameNetModeGet</a></u>	.....	1255
<u><a href="#">axICNSSameNetModeSet</a></u>	.....	1257
<u><a href="#">axICNSSetPhysical</a></u>	.....	1259
<u><a href="#">axICNSSetSpacing</a></u>	.....	1262
<u><a href="#">axICNSSetPinDelayEnabled</a></u>	.....	1266
<u><a href="#">axICNSSetPinDelayPVF</a></u>	.....	1267
<u><a href="#">axICNSSetSameNet</a></u>	.....	1268
<u><a href="#">axICNSSetSameNetXtalkEnabled</a></u>	.....	1270
<u><a href="#">axICNSSetViaZEnabled</a></u>	.....	1271
<u><a href="#">axICNSSetViaZPVF</a></u>	.....	1272
<u><a href="#">axICNSSpacingMax</a></u>	.....	1273
<u><a href="#">axICNSSpacingMin</a></u>	.....	1274
<u><a href="#">axICNSSpacingModeGet</a></u>	.....	1276
<u><a href="#">axICNSSpacingModeSet</a></u>	.....	1278
<u><a href="#">axICnsPurgeAll()</a></u>	.....	1280
<u><a href="#">axICnsPurgeCsets</a></u>	.....	1281
<u><a href="#">axICnsPurgeObjects</a></u>	.....	1282
<u><a href="#">axIViaZLength</a></u>	.....	1283
<u><a href="#">axINetEcsetValueGet</a></u>	.....	1285
<u><a href="#">axICNSEcsetValueSet</a></u>	.....	1287
<u><a href="#">axICnsGetViaList</a></u>	.....	1289
<u><a href="#">axIGetAllViaList</a></u>	.....	1291
<u><a href="#">axIDRCUpdate</a></u>	.....	1292
<u><a href="#">axIDRCWaive</a></u>	.....	1293

## Allegro SKILL Reference

---

<u><a href="#">axIDRCGetCount</a></u> .....	1295
<u><a href="#">axIDRCItem</a></u> .....	1296
<u><a href="#">axIDRCWaiveGetCount</a></u> .....	1298
<u><a href="#">axILayerSet</a></u> .....	1299
<u><a href="#">axICnsList</a></u> .....	1300
<u><a href="#">axICNSMapClear</a></u> .....	1302
<u><a href="#">axICNSMapUpdate</a></u> .....	1303
<u><a href="#">axICnsNetFlattened</a></u> .....	1305
<u><a href="#">DFM DRC Modes Functions</a></u> .....	1307
<u><a href="#">axICNSModeSet</a></u> .....	1307
<u><a href="#">axICNSModeGet</a></u> .....	1310
 <u><a href="#">20</a></u>	
<u><a href="#">Command Control Functions</a></u> .....	1313
<u><a href="#">AXL-SKILL Command Control Functions</a></u> .....	1313
<u><a href="#">axICmdGetAppMode</a></u> .....	1314
<u><a href="#">axICmdIsRegistered</a></u> .....	1315
<u><a href="#">axICmdRegister</a></u> .....	1316
<u><a href="#">axICmdUnregister</a></u> .....	1322
<u><a href="#">axIEndSkillMode</a></u> .....	1323
<u><a href="#">axIFlushDisplay</a></u> .....	1324
<u><a href="#">axIOKToProceed</a></u> .....	1326
<u><a href="#">axISetLineLock</a></u> .....	1327
<u><a href="#">axISetRotateIncrement</a></u> .....	1329
<u><a href="#">axUIGetUserData</a></u> .....	1330
<u><a href="#">axUIPopupDefine</a></u> .....	1331
<u><a href="#">axUIPopupSet</a></u> .....	1333
<u><a href="#">axBuildClassPopup</a></u> .....	1335
<u><a href="#">axBuildSubclassPopup</a></u> .....	1336
<u><a href="#">axSubclassFormPopup</a></u> .....	1338
<u><a href="#">axVisibleUpdate</a></u> .....	1341
 <u><a href="#">21</a></u>	
<u><a href="#">Polygon Operation Functions</a></u> .....	1343
<u><a href="#">About Polygon Operations</a></u> .....	1343

## Allegro SKILL Reference

---

<u>AXL-SKILL Polygon Operation Attributes</u>	1346
<u>AXL-SKILL Polygon Operation Functions</u>	1348
<u>axIPolyFromDB</u>	1348
<u>axIPolyFromHole</u>	1353
<u>axIPolyMemUse</u>	1354
<u>axIPolyOffset</u>	1356
<u>axIPolyOperation</u>	1357
<u>axIPolyExpand</u>	1359
<u>axIPolyType</u>	1365
<u>axIPolyFromDrillHole</u>	1366
<u>axIPolyFromHole</u>	1367
<u>axIPolyErrorGet</u>	1368
<u>axIPolyRotate</u>	1370
<u>axIPolyTrim</u>	1371
<u>Use Models</u>	1372

## 22

<u>Allegro PCB Editor File Access Functions</u>	1375
<u>AXL-SKILL File Access Functions</u>	1375
<u>axIDMFileError</u>	1376
<u>axIDMFindFile</u>	1377
<u>axIDMGetFile</u>	1379
<u>axIDMOpenFile</u>	1381
<u>axIDMOpenLog</u>	1384
<u>axIDMClose</u>	1385
<u>axIDMBrowsePath</u>	1386
<u>axIDMDirectoryBrowse</u>	1387
<u>axIDMFileBrowse</u>	1388
<u>axIDMFileParts</u>	1391
<u>axIOSFileCopy</u>	1392
<u>axIOSFileMove</u>	1393
<u>axIOSSlash</u>	1394
<u>axRecursiveDelete</u>	1395
<u>axTempDirectory</u>	1397
<u>axTempFile</u>	1398

<u><a href="#">axITempFileRemove</a></u>	.....	1399
<b>23</b>		
<u><a href="#">Reports and Extract Functions</a></u>	.....	1401
<u><a href="#">AXL-SKILL Data Extract Functions</a></u>	.....	1401
<u><a href="#">axIExtractToFile</a></u>	.....	1401
<u><a href="#">axIExtractMap</a></u>	.....	1403
<u><a href="#">axIReportGenerate</a></u>	.....	1405
<u><a href="#">axIReportList</a></u>	.....	1407
<u><a href="#">axIReportRegister</a></u>	.....	1408
<b>24</b>		
<u><a href="#">Utility Functions</a></u>	.....	1413
<u><a href="#">axICheckString</a></u>	.....	1414
<u><a href="#">axICmdList</a></u>	.....	1416
<u><a href="#">axIDebug</a></u>	.....	1417
<u><a href="#">axIDetailLoad</a></u>	.....	1418
<u><a href="#">axIDetailSave</a></u>	.....	1420
<u><a href="#">axIEmail</a></u>	.....	1421
<u><a href="#">axIHistory</a></u>	.....	1422
<u><a href="#">axIHttp</a></u>	.....	1424
<u><a href="#">axIIsDebug</a></u>	.....	1426
<u><a href="#">axIIsProductLineActive</a></u>	.....	1427
<u><a href="#">axIISProductStarted</a></u>	.....	1428
<u><a href="#">axILicDefaultVersion</a></u>	.....	1429
<u><a href="#">axILicFeatureExists</a></u>	.....	1430
<u><a href="#">axILicIsProductEnabled</a></u>	.....	1431
<u><a href="#">axILogHeader</a></u>	.....	1432
<u><a href="#">axIMKS2UU</a></u>	.....	1433
<u><a href="#">axIMKSAlias</a></u>	.....	1435
<u><a href="#">axIMKSCConvert</a></u>	.....	1436
<u><a href="#">axIMKSStr2UU</a></u>	.....	1439
<u><a href="#">axIMapClassName</a></u>	.....	1440
<u><a href="#">axIMemSize</a></u>	.....	1442
<u><a href="#">axIOSBackSlash</a></u>	.....	1443

## Allegro SKILL Reference

---

<u><a href="#">axIOSControl</a></u> . . . . .	1444
<u><a href="#">axIOSNtp</a></u> . . . . .	1446
<u><a href="#">axIOSExit</a></u> . . . . .	1448
<u><a href="#">axIPPrint</a></u> . . . . .	1449
<u><a href="#">axIPdfView</a></u> . . . . .	1450
<u><a href="#">axIPrintDbid</a></u> . . . . .	1451
<u><a href="#">axIRegexpls</a></u> . . . . .	1453
<u><a href="#">axIRunBatchDBProgram</a></u> . . . . .	1454
<u><a href="#">axIShowObject</a></u> . . . . .	1459
<u><a href="#">axISleep</a></u> . . . . .	1460
<u><a href="#">axISort</a></u> . . . . .	1461
<u><a href="#">axIstrcmpAlpNum</a></u> . . . . .	1465
<u><a href="#">axIStringCSVParse</a></u> . . . . .	1466
<u><a href="#">axIStringRemoveSpaces</a></u> . . . . .	1468
<u><a href="#">axIVersion</a></u> . . . . .	1469
<u><a href="#">axIVersionIdGet</a></u> . . . . .	1473
<u><a href="#">axIVersionIdPrint</a></u> . . . . .	1474

## 25

<u><a href="#">Math Utility Functions</a></u> . . . . .	1477
<u><a href="#">Overview</a></u> . . . . .	1477
<u><a href="#">axIDegToRad</a></u> . . . . .	1477
<u><a href="#">axIDistance</a></u> . . . . .	1478
<u><a href="#">axIGeo2Str</a></u> . . . . .	1479
<u><a href="#">axIGeoAngleBetweenLines</a></u> . . . . .	1481
<u><a href="#">axIGeoArcCenterAngle</a></u> . . . . .	1482
<u><a href="#">axIGeoArcCenterRadius</a></u> . . . . .	1485
<u><a href="#">axIGeoArcMidpoint</a></u> . . . . .	1490
<u><a href="#">axIGeoCircleCircleInt</a></u> . . . . .	1491
<u><a href="#">axIGeoCircleLineInt</a></u> . . . . .	1492
<u><a href="#">axIGeoCircleLineInt2</a></u> . . . . .	1493
<u><a href="#">axIGeoEqual</a></u> . . . . .	1494
<u><a href="#">axIGeoFindAngle</a></u> . . . . .	1496
<u><a href="#">axIGeoGetBBox</a></u> . . . . .	1497
<u><a href="#">axIGeoIsBoxOverlap</a></u> . . . . .	1498

## Allegro SKILL Reference

---

<u><a href="#">axlGeoLineMidpoint</a></u> . . . . .	1499
<u><a href="#">axlGeoRotatePt</a></u> . . . . .	1500
<u><a href="#">axlGeoPointsEqual</a></u> . . . . .	1502
<u><a href="#">axlGeoPickShorterArc</a></u> . . . . .	1503
<u><a href="#">axlGeolsShorterArcClockwise</a></u> . . . . .	1504
<u><a href="#">axlMathSolveQuadratic</a></u> . . . . .	1505
<u><a href="#">axllsBetween</a></u> . . . . .	1506
<u><a href="#">axllsPointInsideBox</a></u> . . . . .	1507
<u><a href="#">axllsPointOnLine</a></u> . . . . .	1508
<u><a href="#">axlLineSlope</a></u> . . . . .	1509
<u><a href="#">axlLineXLine – Obsolete</a></u> . . . . .	1510
<u><a href="#">axlMathConstants</a></u> . . . . .	1511
<u><a href="#">axlMathDotProduct</a></u> . . . . .	1512
<u><a href="#">axlMidPointArc</a></u> . . . . .	1513
<u><a href="#">axlMidPointLine</a></u> . . . . .	1514
<u><a href="#">axlMPythag</a></u> . . . . .	1515
<u><a href="#">axlMUniVector</a></u> . . . . .	1516
<u><a href="#">axlMXYAdd</a></u> . . . . .	1518
<u><a href="#">axlMXYMult</a></u> . . . . .	1519
<u><a href="#">axlMXYSub</a></u> . . . . .	1520
<u><a href="#">axlRadToDeg</a></u> . . . . .	1521
<u><a href="#">axl ol ol2</a></u> . . . . .	1522
<u><a href="#">bBoxAdd</a></u> . . . . .	1524

## 26

<u><a href="#">Database Miscellaneous Functions</a></u> . . . . .	1525
<u><a href="#">Overview</a></u> . . . . .	1525
<u><a href="#">axlAirGap</a></u> . . . . .	1526
<u><a href="#">axlBackDrill – Obsolete</a></u> . . . . .	1531
<u><a href="#">axlDBGetLength</a></u> . . . . .	1532
<u><a href="#">axlDBGetManhattan</a></u> . . . . .	1533
<u><a href="#">axlDBGetSymbolBodyExtent</a></u> . . . . .	1534
<u><a href="#">axlDBPinPairLength</a></u> . . . . .	1535
<u><a href="#">axlDeleteByLayer</a></u> . . . . .	1536
<u><a href="#">axlExtentDB</a></u> . . . . .	1538

## Allegro SKILL Reference

---

<u><a href="#">axlExtentLayout – Obsolete</a></u>	.....	1539
<u><a href="#">axlExtentSymbol – Obsolete</a></u>	.....	1540
<u><a href="#">axlFindPath</a></u>	.....	1541
<u><a href="#">axlGeoClosestPointOnArc</a></u>	.....	1543
<u><a href="#">axlGeoClosestPointOnCircle</a></u>	.....	1544
<u><a href="#">axlGeoPointInShape</a></u>	.....	1545
<u><a href="#">axlGeoPointShapeInfo</a></u>	.....	1546
<u><a href="#">axlGetImpedance</a></u>	.....	1547
<u><a href="#">axlGeoMirrorLayer</a></u>	.....	1548
<u><a href="#">axlImpdedanceGetLayerBroadsideDPImp</a></u>	.....	1549
<u><a href="#">axlImpdedanceGetLayerBroadsideDPWidth</a></u>	.....	1550
<u><a href="#">axlImpdedanceGetLayerEdgeDPImp</a></u>	.....	1551
<u><a href="#">axlImpdedanceGetLayerEdgeDPSpacing</a></u>	.....	1552
<u><a href="#">axlImpdedanceGetLayerEdgeDPWidth</a></u>	.....	1553
<u><a href="#">axlImpedance2Width</a></u>	.....	1554
<u><a href="#">axlPadOnLayer</a></u>	.....	1555
<u><a href="#">axlPinExport</a></u>	.....	1557
<u><a href="#">axlPinImport</a></u>	.....	1558
<u><a href="#">axlReratNet</a></u>	.....	1560
<u><a href="#">axlText2Lines</a></u>	.....	1561
<u><a href="#">axlUnfixAll</a></u>	.....	1563
<u><a href="#">axlWidth2Impedance</a></u>	.....	1564
<u><a href="#">axlIsHighlighted</a></u>	.....	1565
<u><a href="#">axlTestPoint</a></u>	.....	1566
<u><a href="#">axlChangeNet</a></u>	.....	1569
<u><a href="#">axlSegDelayAndZ0</a></u>	.....	1571
<u><a href="#">axlSetDefaultDielInformation</a></u>	.....	1572

## 27

<u><a href="#">Microsoft Excel Integration Functions</a></u>	.....	1573
<u><a href="#">axlSpreadsheetClose</a></u>	.....	1573
<u><a href="#">axlSpreadsheetDefineCell</a></u>	.....	1575
<u><a href="#">axlSpreadsheetDoc</a></u>	.....	1576
<u><a href="#">axlSpreadsheetGetCell</a></u>	.....	1578
<u><a href="#">axlSpreadsheetGetRGBColorString</a></u>	.....	1580

## Allegro SKILL Reference

---

<u><a href="#">axISpreadsheetGetRGBForNamedColor</a></u> . . . . .	1581
<u><a href="#">axISpreadsheetGetStyles</a></u> . . . . .	1582
<u><a href="#">axISpreadsheetGetWorksheets</a></u> . . . . .	1583
<u><a href="#">axISpreadsheetGetWorksheetSize</a></u> . . . . .	1584
<u><a href="#">axISpreadsheetInit</a></u> . . . . .	1585
<u><a href="#">axISpreadsheetRead</a></u> . . . . .	1587
<u><a href="#">axISpreadsheetReadDelimited</a></u> . . . . .	1589
<u><a href="#">axISpreadsheetSetCell</a></u> . . . . .	1590
<u><a href="#">axISpreadsheetSetCellProp</a></u> . . . . .	1592
<u><a href="#">axISpreadsheetSetColumnProp</a></u> . . . . .	1594
<u><a href="#">axISpreadsheetSetDocProp</a></u> . . . . .	1596
<u><a href="#">axISpreadsheetSetRowProp</a></u> . . . . .	1597
<u><a href="#">axISpreadsheetSetStyle</a></u> . . . . .	1599
<u><a href="#">axISpreadsheetSetStyleBorder</a></u> . . . . .	1600
<u><a href="#">axISpreadsheetSetStyleParent</a></u> . . . . .	1602
<u><a href="#">axISpreadsheetSetStyleProp</a></u> . . . . .	1603
<u><a href="#">axISpreadsheetSetWorksheet</a></u> . . . . .	1607
<u><a href="#">axISpreadsheetWrite</a></u> . . . . .	1608

## 28

<u><a href="#">Plugin Functions</a></u> . . . . .	1611
<u><a href="#">Overview</a></u> . . . . .	1611
<u><a href="#">SKILL Programming</a></u> . . . . .	1611
<u><a href="#">DLL Programming</a></u> . . . . .	1612
<u><a href="#">Input/Output Data Primitives</a></u> . . . . .	1614
<u><a href="#">Programming Restrictions, Cautions and Hints</a></u> . . . . .	1615
<u><a href="#">Performance Considerations</a></u> . . . . .	1616
<u><a href="#">Cadence Customer Support</a></u> . . . . .	1616
<u><a href="#">Examples</a></u> . . . . .	1616
<u><a href="#">axIDllCall</a></u> . . . . .	1618
<u><a href="#">axIDllCallList</a></u> . . . . .	1620
<u><a href="#">axIDllClose</a></u> . . . . .	1621
<u><a href="#">axIDllDump</a></u> . . . . .	1622
<u><a href="#">axIDllOpen</a></u> . . . . .	1623
<u><a href="#">axIDllSym</a></u> . . . . .	1625

### 29

<u>Skill Language Extensions</u> .....	1627
<u>axldo</u> .....	1627
<u>copyDeep</u> .....	1629
<u>isBoxp</u> .....	1630
<u>lastelem</u> .....	1631
<u>letStar</u> .....	1632
<u>listnindex</u> .....	1633
<u>movedown</u> .....	1634
<u>moveup</u> .....	1635
<u>parseFile</u> .....	1636
<u>parseQuotedString</u> .....	1638
<u>pprintIn</u> .....	1639
<u>propNames</u> .....	1640

### 30

<u>Logic Access Functions</u> .....	1641
<u>Overview</u> .....	1641
<u>axIDBAssignNet</u> .....	1642
<u>axIDBCreateConceptComponent</u> .....	1645
<u>axIDBCreateComponent</u> .....	1647
<u>axIDBCreateManyModuleInstances</u> .....	1649
<u>axIDBCreateModuleDef</u> .....	1652
<u>axIDBCreateModuleInstance</u> .....	1653
<u>axIDBCreateNet</u> .....	1655
<u>axIDBCreateSymDefSkeleton</u> .....	1656
<u>axIDBDummyNet</u> .....	1658
<u>axIdbidName</u> .....	1660
<u>axIDiffPair</u> .....	1662
<u>axIDiffPairAuto</u> .....	1664
<u>axIDiffPairDBID</u> .....	1666
<u>axIMatchGroupAdd</u> .....	1667
<u>axIMatchGroupCreate</u> .....	1669
<u>axIMatchGroupDelete</u> .....	1672

## Allegro SKILL Reference

---

<u><a href="#">axIMatchGroupProp</a></u> . . . . .	1673
<u><a href="#">axIMatchGroupRemove</a></u> . . . . .	1675
<u><a href="#">axINetSched</a></u> . . . . .	1677
<u><a href="#">axIPinPair</a></u> . . . . .	1678
<u><a href="#">axIPinPairSeek</a></u> . . . . .	1681
<u><a href="#">axIPinsOfNet</a></u> . . . . .	1682
<u><a href="#">axIRemoveNet</a></u> . . . . .	1683
<u><a href="#">axIRenameNet</a></u> . . . . .	1684
<u><a href="#">axIRenameRefdes</a></u> . . . . .	1686
<u><a href="#">axISchedule</a></u> . . . . .	1688
<u><a href="#">axIScheduleNet</a></u> . . . . .	1690
<u><a href="#">axIWriteDeviceFile</a></u> . . . . .	1691
<u><a href="#">axIWritePackageFile</a></u> . . . . .	1693

## 31

<u><a href="#">DFM SKILL APIs</a></u> . . . . .	1695
<u><a href="#">Component Lead Functions</a></u> . . . . .	1695
<u><a href="#">axICreateLeadContact</a></u> . . . . .	1695
<u><a href="#">axIDeleteLeadContact</a></u> . . . . .	1698
<u><a href="#">axIGetSupportedLeadTypes</a></u> . . . . .	1699
<u><a href="#">DFM CSet Operations</a></u> . . . . .	1700
<u><a href="#">axICreateDfmAssignment</a></u> . . . . .	1703
<u><a href="#">axIDFMIscsetLocked</a></u> . . . . .	1706
<u><a href="#">axIDFMCsetLock</a></u> . . . . .	1707
<u><a href="#">axIDeleteDfmAssignment</a></u> . . . . .	1708
<u><a href="#">axIGetDFMAssignedCsets</a></u> . . . . .	1709
<u><a href="#">axIGetDFMCsetAssignmentStatus</a></u> . . . . .	1713
<u><a href="#">axIRenameDFFCset</a></u> . . . . .	1716
<u><a href="#">axICreateDFMCset</a></u> . . . . .	1717
<u><a href="#">axIDeleteDFMCset</a></u> . . . . .	1719
<u><a href="#">axIGetDFMCsets</a></u> . . . . .	1720
<u><a href="#">axIGetDFMCset</a></u> . . . . .	1721

## **Allegro SKILL Reference**

---

# **Alphabetical List of Functions**

---

axl\_ol\_ol2 1522  
axlAddAutoAssignNetAlgorithm 622  
axlAddSelectAll 291  
axlAddSelectBox 289  
axlAddSelectName 296  
axlAddSelectObject 301  
axlAddSelectPoint 283  
axlAddSimpleMoveDynamics 510  
axlAddSimpleRbandDynamics 507  
axlAddTaper 411  
axlAirGap 1526  
axlAllShapesMerge 342  
axlAltSymbolList 446  
axlAltSymbolOK 447  
axlAltSymbolReplace 448  
axlAutoOpenFindFilter 322  
axlBackDrill – Obsolete 1531  
axlBackdrillGet 450  
axlBondFingerDelete 343  
axlBondWireDelete 344  
axlBuildClassPopup 1335  
axlBuildSubclassPopup 1336  
axlCancelClearFormClosable 677  
axlCancelEnterFun 516  
axlCancelOff 673  
axlCancelOn 674  
axlCancelSetFormClosable 678  
axlCancelTest 676  
axlCDLNetlistSetValue 661  
axlChangeLayer 592  
axlChangeLine2Cline 345  
axlChangeLineFont 346  
axlChangeNet 1569  
axlChangeWidth 347  
axlCheckString 1414  
axlClasses 213  
axlClearDynamics 506  
axlClearObjectCustomColor 201  
axlClearSelSet 309

axIClipboardGetText 680  
axIClipboardSetText 681  
axICloseFindFilter 324  
axICmdGetAppMode 1314  
axICmdIsRegistered 1315  
axICmdList 1416  
axICmdRegister 1316  
axICmdUnregister 1322  
axICnsAddVia 1180  
axICNSAssemblyModeGet 594  
axICNSAssemblyModeSet 596  
axICnsAssignPurge – Obsolete 1181  
axICnsClassTableChange 1182  
axICnsClassTableCreate 1184  
axICnsClassTableDelete 1186  
axICnsClassTableFind 1187  
axICnsClassTableSeek 1189  
axICNSCreate 1191  
axICNSCsetLock 1193  
axICNSDelete 1195  
axICnsDeleteClassClassObjects 1196  
axICnsDeleteRegionClassClassObjects 1197  
axICnsDeleteRegionClassObjects 1198  
axICnsDeleteVia 1199  
axICNSDesignModeGet 1201  
axICNSDesignModeSet 1204  
axICNSDesignValueCheck 1207  
axICNSDesignValueGet 1208  
axICNSDesignValueSet 1211  
axICNSDFAExport 1213  
axICNSDFAImport 1214  
axICNSDFAMode 1215  
axICNSEcsetCreate 1217  
axICNSEcsetDelete 1219  
axICNSEcsetGet 1220  
axICNSEcsetModeGet 1221  
axICNSEcsetModeSet 1223  
axICNSEcsetValueCheck 1226  
axICNSEcsetValueGet 1227  
axICNSEcsetValueSet 1287  
axICNSGetAssembly 598  
axICNSGetDefaultMinLineWidth 1230  
axICNSGetPhysical 1231  
axICNSGetPinDelayEnabled 1234

## Allegro SKILL Reference

---

axICNSGetPinDelayPVF 1235  
axICNSGetSameNet 1236  
axICNSGetSameNetXtalkEnabled 1239  
axICNSGetSpacing 1240  
axICnsGetViaList 1289  
axICNSGetViaZEnabled 1243  
axICNSGetViaZPVF 1244  
axICNSIsCsetLocked 1247  
axICNSIsLockedDomain 1248  
axICnsList 1300  
axICNSLockDomain 1250  
axICNSMapClear 1302  
axICNSMapUpdate 1303  
axICNSModeGet 1310  
axICNSModeSet 1307  
axICnsNetFlattened 1305  
axICNSOptions 1251  
axICNSPhysicalModeGet 1245  
axICNSPhysicalModeSet 1253  
axICnsPurgeAll() 1280  
axICnsPurgeCsets 1281  
axICnsPurgeObjects 1282  
axICNSSameNetModeGet 1255  
axICNSSameNetModeSet 1257  
axICNSSetAssembly 601  
axICNSSetPhysical 1259  
axICNSSetPinDelayEnabled 1266  
axICNSSetPinDelayPVF 1267  
axICNSSetSameNet 1268  
axICNSSetSameNetXtalkEnabled 1270  
axICNSSetSpacing 1262  
axICNSSetViaZEnabled 1271  
axICNSSetViaZPVF 1272  
axICNSSpacingMax 1273  
axICNSSpacingMin 1274  
axICNSSpacingModeGet 1276  
axICNSSpacingModeSet 1278  
axIColorDoc 181  
axIColorGet 184  
axIColorLoad 190  
axIColorOnGet – Obsolete 192  
axIColorOnSet – Obsolete 193  
axIColorPriorityGet – Obsolete 194  
axIColorPrioritySet – Obsolete 195

axIColorSave 196  
axIColorSet 197  
axIColorShadowGet 186  
axIColorShadowSet 188  
axICompAddPin 605  
axICompareLayers 214  
axICompDeletePin 608  
axICompileSymbol 965  
axICompMovePin 609  
axIComponentChangeClass 611  
axICompSetPinAttributes 613  
axIConductorBottomLayer 241  
axIConductorTopLayer 242  
axIControlRaise 534  
axICopyObject 351  
axICopyProperties 349  
axIcreate – Obsolete 132  
axICreateAttachment 1156  
axICreateBondFinger 1078  
axICreateBondWire 1080  
axICreateDeviceFileTemplate 604  
axICreateDfmAssignment 1703  
axICreateDFMCset 1717  
axICreateLeadContact 1695  
axICreateWirebondGuide 1110  
axICurrentDesign 962  
axICursorGet 682  
axICursorWarp 683  
axICustomColorObject 203  
axICVFCColorChooserDlg 200  
axIDB2Path 1047  
axIDBActiveShape 1067  
axIDBAddGroupObjects 1130  
axIDBAddProp 1117  
axIDBAltOrigin 353  
axIDBAssignNet 1642  
axIDBChangeDesignExtents 993  
axIDBChangeDesignOrigin 994  
axIDBChangeDesignUnits 995  
axIDBChangeText 355  
axIDBCheck 998  
axIDBCloak 1167  
axIDBComposeShapesFromLines 1056  
axIDBConnectItem 358

axIDBControl 967  
axIDBCopyPadstack 1000  
axIDBCreateBoundingShape 1059  
axIDBCreateCircle 1053  
axIDBCreateCloseShape 1066  
axIDBCreateComponent 1647  
axIDBCreateConceptComponent 1645  
axIDBCreateExternalDRC 1082  
axIDBCreateFillet 1086  
axIDBCreateFilmRec – Obsolete 243  
axIDBCreateGroup 1131  
axIDBCreateLine 1051  
axIDBCreateManyModuleInstances 1649  
axIDBCreateModuleDef 1652  
axIDBCreateModuleInstance 1653  
axIDBCreateNet 1655  
axIDBCreateOpenShape 1062  
axIDBCreatePadStack 420  
axIDBCreatePath 1048  
axIDBCreatePin 1088  
axIDBCreatePropDictEntry 1113  
axIDBCreateRectangle 1073  
axIDBCreateShape 1071  
axIDBCreateSymbol 1092  
axIDBCreateSymbolAutosilk 1107  
axIDBCreateSymbolSkeleton 1096  
axIDBCreateSymDefSkeleton 1656  
axIDBCreateText 1101  
axIDBCreateVectorizedEllipse 1075  
axIDBCreateVia 1104  
axIDBCreateViaStructure 1108  
axIDBCreateVoid 1069  
axIDBCreateVoidCircle 1068  
axIDBDeleteProp 366  
axIDBDeletePropAll 369  
axIDBDeletePropDictEntry 370  
axIDBDelLock 1002  
axIDBDisbandGroup 1135  
axIDBDisplayControl 1022  
axIDBDummyNet 1658  
axIDBDynamicShapes 489  
axIDBFindByName 325  
axIDBGetAttachedText 458  
axIDBGetConnect 472

axIDBGetDesign 453  
axIDBGetDesignUnits 468  
axIDBGetDrillPlating 455  
axIDBGetExtents 974  
axIDBGetGroupFromItem 1136  
axIDBGetLayerType 216  
axIDBGetLength 1532  
axIDBGetLock 1003  
axIDBGetLonelyBranches 471  
axIDBGetManhattan 1533  
axIDBGetPad 460  
axIDBGetPropDict 463  
axIDBGetPropDictEntry 464  
axIDBGetProperties 466  
axIDBGetShapes 490  
axIDBGetSymbolBodyExtent 1534  
axIDBGetTextBlockCount 133  
axIDBGridGet 134  
axIDBGridSet 136  
axIDBGroupRename 1138  
axIDbidName 1660  
axIDBIgnoreFixed 975  
axIDBIsBondingWireLayer – Obsolete 615  
axIDBIsBondpad 474  
axIDBIsBondwire 475  
axIDBIsDiePad 616  
axIDBIsDieStackLayer 632  
axIDBIsFixed 476  
axIDBIsPackagePin 478  
axIDBIsPlatingbarPin 617  
axIDBIsReadOnly 977  
axIDBMemoryReclaim 1004  
axIDBOpenShape 371  
axIDBPinPairLength 1535  
axIDBRefreshId 469  
axIDBRemoveGroupObjects 1139  
axIDBSectorSize - Obsolete 978  
axIDBSetLock 1006  
axIDBTextBlockCompact 491  
axIDBTextBlockCreate 138  
axIDBTextBlockFindName 140  
axIDBTextBlockGetName 141  
axIDBTextBlockSetName 142  
axIDBTransactionCommit 1171

axIDBTransactionMark 1172  
axIDBTransactionOops 1173  
axIDBTransactionRollback 1174  
axIDBTransactionStart 1175  
axIDBTuneSectorSize 1009  
axIDBViaStack 479  
axIDebug 1417  
axIDegToRad 1477  
axIDehighlightObject 526  
axIDeleteAttachment 1159  
axIDeleteByLayer 1536  
axIDeleteByRectangle 363  
axIDeleteDfmAssignment 1708  
axIDeleteDFMCset 1719  
axIDeleteFillet 373  
axIDeleteLeadContact 1698  
axIDeleteObject 360  
axIDeleteTaper 365  
axIDesignFlip 512  
axIDesignType 963  
axIDetailLoad 1418  
axIDetailSave 1420  
axIDFMCsetLock 1707  
axIDFMIIsCsetLocked 1706  
axIDiffPair 1662  
axIDiffPairAuto 1664  
axIDiffPairDBID 1666  
axIDistance 1478  
axIDIIICall 1618  
axIDIIICallList 1620  
axIDIIClose 1621  
axIDIIDump 1622  
axIDIIOpen 1623  
axIDIISym 1625  
axIDMBrowsePath 1386  
axIDMClose 1385  
axIDMDirectoryBrowse 1387  
axIDMFileBrowse 1388  
axIDMFileError 1376  
axIDMFileParts 1391  
axIDMFindFile 1377  
axIDMGetFile 1379  
axIDMOpenFile 1381  
axIDMOpenLog 1384

axldo 1627  
axlDrawObject 531  
axlDRCGetCount 1295  
axlDRCItem 1296  
axlDRCUpdate 1292  
axlDRCWaive 1293  
axlDRCWaiveGetCount 1298  
axlDynamicsObject 532  
axlEmail 1421  
axlEndSkillMode 1323  
axlEnterAngle 515  
axlEnterBox 521  
axlEnterEvent 535  
axlEnterPath 523  
axlEnterPoint 513  
axlEnterString 514  
axlEraseObject 533  
axlEventSetStartPopup 539  
axlExportXmlIDBRecords 143  
axlExtentDB 1538  
axlExtentLayout – Obsolete 1539  
axlExtentSymbol – Obsolete 1540  
axlExtractMap 1403  
axlExtractToFile 1401  
axlFillet 374  
axlFilletConvert 375  
axlFilmCreate 144  
axlFindFilterIsOpen 327  
axlFindPath 1541  
axlFinishEnterFun 517  
axlFlushDisplay 1324  
axlFormAutoResize 863  
axlFormBNFDoc 798  
axlFormBuildPopup 825  
axlFormCallback 814  
axlFormClearMouseActive 822  
axlFormClose 823  
axlFormColorize 864  
axlFormCreate 819  
axlFormDefaultButton 903  
axlFormDisplay 824  
axlFormGetActiveField 867  
axlFormGetField 829  
axlFormGetFieldRange 831

axlFormGetFieldRange 901  
axlFormGetFieldType 902  
axlFormGetOptionValue 832  
axlFormGridBatch 868  
axlFormGridCancelPopup 869  
axlFormGridDeleteRows 870  
axlFormGridEvents 871  
axlFormGridGetCell 874  
axlFormGridGetSize 833  
axlFormGridInsertCol 876  
axlFormGridInsertRows 880  
axlFormGridNewCell 881  
axlFormGridOptions 905  
axlFormGridReset 882  
axlFormGridSelected 835  
axlFormGridSelectedCnt 836  
axlFormGridSetBatch 883  
axlFormGridSetSelectRows 837  
axlFormGridUpdate 886  
axlFormInvalidateField 887  
axlFormIsFieldEditable 888  
axlFormIsFieldVisible 857  
axlFormListAddItem 889  
axlFormListDeleteAll 839  
axlFormListDeleteItem 891  
axlFormListGetCount 834  
axlFormListGetItem 893  
axlFormListGetSelCount 894  
axlFormListGetSelItems 895  
axlFormListOptions 896  
axlFormListSelAll 898  
axlFormListSelect 842  
axlFormMsg 899  
axlFormRestoreField 852  
axlFormSetActiveField 907  
axlFormSetDecimal 908  
axlFormSetEventAction 843  
axlFormSetField 845  
axlFormSetFieldEditable 909  
axlFormSetFieldHelpTip 848  
axlFormSetFieldLimits 910  
axlFormSetFieldVisible 856  
axlFormSetInfo 849  
axlFormSetMouseActive 850

axlFormTest 851  
axlFormTitle 854  
axlFormTreeViewAddItem 911  
axlFormTreeViewChangeImages 914  
axlFormTreeViewChangeLabel 916  
axlFormTreeViewGetImages 917  
axlFormTreeViewGetLabel 918  
axlFormTreeViewGetParents 919  
axlFormTreeViewGetSelectState 920  
axlFormTreeViewLoadBitmaps 921  
axlFormTreeViewSet 923  
axlFormTreeViewSetSelectState 926  
axlGeo2Str 1479  
axlGeoAngleBetweenLines 1481  
axlGeoArcCenterAngle 1482  
axlGeoArcCenterRadius 1485  
axlGeoArcMidpoint 1490  
axlGeoCircleCircleInt 1491  
axlGeoCircleLineInt 1492  
axlGeoCircleLineInt2 1493  
axlGeoClosestPointOnArc 1543  
axlGeoClosestPointOnCircle 1544  
axlGeoEqual 1494  
axlGeoFindAngle 1496  
axlGeoGetBBox 1497  
axlGeoIsBoxOverlap 1498  
axlGeoIsShorterArcClockwise 1504  
axlGeoLineMidpoint 1499  
axlGeoMirrorLayer 1548  
axlGeoPickShorterArc 1503  
axlGeoPointInShape 1545  
axlGeoPointsEqual 1502  
axlGeoPointShapeInfo 1546  
axlGeoRotatePt 1500  
axlGetActiveLayer – Obsolete 1018  
axlGetActiveTextBlock – Obsolete 1019  
axlGetAlias 564  
axlGetAllAttachmentNames 1160  
axlGetAllViaList 1291  
axlGetAllVisibleProfiles 624  
axlGetAttachment 1161  
axlGetBondWireLength 626  
axlGetDFMAssignedCsets 1709  
axlGetDFMCset 1721

axIGetDFMCsetAssignmentStatus 1713  
axIGetDFMCsets 1720  
axIGetDieData 633  
axIGetDieStackData 636  
axIGetDieStackMemberSet 638  
axIGetDieStackNames 640  
axIGetDieType 618  
axIGetDrawingName 979  
axIGetDynamicsSegs 518  
axIGetExternalLayers 217  
axIGetFindFilter 310  
axIGetFunkey 565  
axIGetImpedance 1547  
axIGetIposerData 641  
axIGetLastEnterPoint 376  
axIGetLineLock 519  
axIGetMetalUsageForLayer 619  
axIGetModuleInstanceDefinition 480  
axIGetModuleInstanceLocation 481  
axIGetModuleInstanceLogicMethod 482  
axIGetModuleInstanceNetExceptions 483  
axIGetParam 168  
axIGetSelSet 306  
axIGetSelSetCount 308  
axIGetSpacerData 643  
axIGetSupportedLeadTypes 1699  
axIGetTrapBox 541  
axIGetVariable 566  
axIGetVariableList 568  
axIGetWireProfileColor 645  
axIGetWireProfileDefinition 621  
axIGetWireProfileDirection 623  
axIGetWireProfileVisible 646  
axIGetXSection – Obsolete 218  
axIGRPDrwBitmap 933  
axIGRPDrwCircle 934  
axIGRPDrwInit 935  
axIGRPDrwLine 936  
axIGRPDrwMapWindow 937  
axIGRPDrwPoly 938  
axIGRPDrwRectangle 939  
axIGRPDrwText 940  
axIGRPDrwUpdate 941  
axIHighlightObject 524

axIHISTORY 1422  
axIHttp 1424  
axIgnoreFixed 980  
axImpedanceGetLayerBroadsideDPImp 1549  
axImpedanceGetLayerBroadsideDPWidth 1550  
axImpedanceGetLayerEdgeDPImp 1551  
axImpedanceGetLayerEdgeDPSpacing 1552  
axImpedanceGetLayerEdgeDPWidth 1553  
axImpedance2Width 1554  
axImportWireProfileDefinitions 629  
axImportXmlDBRecords 147  
axInTrigger 981  
axInTriggerFunc 982  
axIsAttachment 1163  
axIsBetween 1506  
axIsCustomColored 212  
axIsDBIDType 456  
axIsDebug 1426  
axIsDummyNet 484  
axIsEtchLayer 220  
axIsFormType 855  
axIsGridCellType 879  
axIsHighlighted 1565  
axIsitFill 487  
axIsLayer 221  
axIsLayerNegative 485  
axIsParamType 177  
axIsPinUnused 486  
axIsPointInsideBox 1507  
axIsPointOnLine 1508  
axIsPolyType 1365  
axIsProductLineActive 1427  
axISProductStarted 1428  
axIsProtectAlias 573  
axIsSymbolEditor 983  
axIsViewFileType 721  
axIsVisibleLayer 222  
axJournal 570  
axKillDesign 984  
axLastPick 377  
axLastPickIsSnapped 339  
axLayerCreateCrossSection – Obsolete 223  
axLayerCreateNonConductor 224  
axLayerDelete 225

axILayerExternal 227  
axILayerGet 228  
axILayerPriorityClearAll 205  
axILayerPriorityGet 206  
axILayerPriorityRestoreAll 208  
axILayerPrioritySaveAll 209  
axILayerPrioritySet 210  
axILayerRenameNonConductor 230  
axILayerSet 1299  
axILayerViaLabel 231  
axILicDefaultVersion 1429  
axILicFeatureExists 1430  
axILicIsProductEnabled 1431  
axILineSlope 1509  
axILineXLine – Obsolete 1510  
axILoadPadstack 1122  
axILoadSymbol 1123  
axILoadViaStructure 627  
axILogHeader 1432  
axIMakeDynamicsPath 559  
axIMapClassName 1440  
axIMatchGroupAdd 1667  
axIMatchGroupCreate 1669  
axIMatchGroupDelete 1672  
axIMatchGroupProp 1673  
axIMatchGroupRemove 1675  
axIMaterialGet 232  
axIMathConstants 1511  
axIMathDotProduct 1512  
axIMathSolveQuadratic 1505  
axIMemSize 1442  
axIMeterCreate 684  
axIMeterDestroy 686  
axIMeterIsCancelled 687  
axIMeterUpdate 688  
axIMidPointArc 1513  
axIMidPointLine 1514  
axIMiniStatusClose 530  
axIMiniStatusLoad 527  
axIMiniStatusReset 149  
axIMKS2UU 1433  
axIMKSAlias 1435  
axIMKSConvert 1436  
axIMKSStr2UU 1439

axIMPythag 1515  
axIMsgCancelPrint 956  
axIMsgCancelSeen 957  
axIMsgClear 958  
axIMsgContextClear 955  
axIMsgContextFinish 954  
axIMsgContextGet 949  
axIMsgContextGetString 948  
axIMsgContextInBuf 951  
axIMsgContextPrint 947  
axIMsgContextRemove 952  
axIMsgContextStart 953  
axIMsgContextTest 950  
axIMsgPut 946  
axIMsgSet 959  
axIMsgTest 960  
axIMUniVector 1516  
axIMXYAdd 1518  
axIMXYSMult 1519  
axIMXYSSub 1520  
axINetClassAdd 1140  
axINetClassCreate 1142  
axINetClassDelete 1144  
axINetClassGet 1145  
axINetClassRemove 1147  
axINetEcsetValueGet 1285  
axINetSched 1677  
axIOK2Void 488  
axIOKToProceed 1326  
axOpenDesign 985  
axOpenDesignForBatch 987  
axOpenFindFilter 323  
axIOSBackSlash 1443  
axIOSControl 1444  
axOSExit 1448  
axIOSFileCopy 1392  
axIOSFileMove 1393  
axOSNtp 1446  
axOSSlash 1394  
axIPackageDesignCheckAddCategory 647  
axIPackageDesignCheckAddCheck 649  
axIPackageDesignCheckDrcError 652  
axIPackageDesignCheckLogError 653  
axIPadFigureTypes 428

axIPadOnLayer 1555  
axIPadstackEdit 429  
axIPadstackSetType 439  
axIPadstackToDisk 1125  
axIPadstackUsageTypes 441  
axIPadSuppressGet 150  
axIPadSuppressOkLayer 152  
axIPadSuppressSet 153  
axIPadUserMaskLayers 442  
axIParamDesignDoc 173  
axIParamFilletDoc 156  
axIParamShapeDoc 160  
axIPathArcAngle 1033  
axIPathArcCenter 1033  
axIPathArcRadius 1033  
axIPathGetLastPathSeg 1041  
axIPathGetPathSegs 1040  
axIPathGetWidth 1038  
axIPathLine 1037  
axIPathOffset 1046  
axIPathSegGetArcCenter 1043  
axIPathSegGetArcClockwise 1044  
axIPathSegGetEndPoint 1042  
axIPathSegGetWidth 1039  
axIPathStart 1032  
axIPathStartCircle 1045  
axIPdfView 1450  
axIPinExport 1557  
axIPinImport 1558  
axIPinPair 1678  
axIPinPairSeek 1681  
axIPinsOfNet 1682  
axIPolyErrorGet 1368  
axIPolyExpand 1359  
axIPolyFromDB 1348  
axIPolyFromDrillHole 1366  
axIPolyFromHole 1353  
axIPolyFromHole 1367  
axIPolyMemUse 1354  
axIPolyOffset 1356  
axIPolyOperation 1357  
axIPolyRotate 1370  
axIPolyTrim 1371  
axIPPrint 1449

## Allegro SKILL Reference

---

axlPrintDbid 1451  
axlProtectAlias 572  
axlPurge3DModelMapDataInDesign 383  
axlPurgePadstacks 381  
axlRadToDeg 1521  
axlRatsnestBlank 542  
axlRatsnestDisplay 543  
axlReadOnlyVariable 574  
axlRecursiveDelete 1395  
axlRefreshSymbol 1126  
axlRegexpls 1453  
axlRegionAdd 1148  
axlRegionCreate 1150  
axlRegionDelete 1151  
axlRegionRemove 1152  
axlRemoveNet 1683  
axlRenameDesign 989  
axlRenameDFFCset 1716  
axlRenameNet 1684  
axlRenameRefdes 1686  
axlReplacePadstack 380  
axlReportGenerate 1405  
axlReportList 1407  
axlReportRegister 1408  
axlReratNet 1560  
axlRunBatchDBProgram 1454  
axlSaveDesign 990  
axlSaveEnable 992  
axlSchedule 1688  
axlScheduleNet 1690  
axlSegDelayAndZ0 1571  
axlSelect 303  
axlSelectByName 328  
axlSelectByProperty 333  
axlSetActiveLayer – Obsolete 1020  
axlSetAlias 576  
axlSetAlias 578  
axlSetAllProfilesVisible 625  
axlSetAttachment 1164  
axlSetBondWireProfile 628  
axlSetBondWireProfile 630  
axlSetDefaultDieInformation 1572  
axlSetDieData 655  
axlSetDieStackData 631

axISetDieType 656  
axISetDynamicsMirror 544  
axISetDynamicsRotation 545  
axISetFindFilter 312  
axISetFunkey 580  
axISetIposerData 657  
axISetLineLock 1327  
axISetParam 175  
axISetPlaneType 244  
axISetRotateIncrement 1329  
axISetSpacerData 658  
axISetSymbolType 966  
axISetVariable 582  
axISetVariableFile 584  
axISetWireProfileColor 659  
axISetWireProfileVisible 660  
axIShapeArea 492  
axIShapeAutoVoid 384  
axIShapeChangeDynamicType 386  
axIShapeDeleteVoids 388  
axIShapeDynamicUpdate 390  
axIShapeMerge 393  
axIShapeRaisePriority 391  
axIShell 585  
axIShellPost 586  
axIShoeItems 395  
axIShoeSetParams 396  
axIShowObject 1459  
axIShowObjectToFile 546  
axISingleSelectBox 287  
axISingleSelectName 294  
axISingleSelectObject 300  
axISingleSelectPoint 280  
axISleep 1460  
axISmoothDesign 399  
axISmoothItems 400  
axISmoothSetParams 401  
axISnapDisableAtRMB 335  
axISnapEnableAtRMB 336  
axISnapToObject 337  
axISort 1461  
axISpreadsheetClose 1573  
axISpreadsheetDefineCell 1575  
axISpreadsheetDoc 1576

## Allegro SKILL Reference

---

axISpreadsheetGetCell 1578  
axISpreadsheetGetRGBColorString 1580  
axISpreadsheetGetRGBForNamedColor 1581  
axISpreadsheetGetStyles 1582  
axISpreadsheetGetWorksheets 1583  
axISpreadsheetGetWorksheetSize 1584  
axISpreadsheetInit 1585  
axISpreadsheetRead 1587  
axISpreadsheetReadDelimited 1589  
axISpreadsheetSetCell 1590  
axISpreadsheetSetCellProp 1592  
axISpreadsheetSetColumnProp 1594  
axISpreadsheetSetDocProp 1596  
axISpreadsheetSetRowProp 1597  
axISpreadsheetSetStyle 1599  
axISpreadsheetSetStyleBorder 1600  
axISpreadsheetSetStyleParent 1602  
axISpreadsheetSetStyleProp 1603  
axISpreadsheetSetWorksheet 1607  
axISpreadsheetWrite 1608  
axIStepDelete 404  
axIStepGet 493  
axIStepMappedInstance 496  
axIStepSet 405  
axIStrcmpAlpNum 1465  
axIStringCSVParse 1466  
axIStringRemoveSpaces 1468  
axISubclasses 245  
axISubclassFormPopup 1338  
axISubclassRoute 247  
axISubSelectAll 292  
axISubSelectBox 290  
axISubSelectName 298  
axISubSelectObject 302  
axISubSelectPoint 285  
axISymbolAttach 407  
axISymbolDetach 409  
axItdfIsEnabled 178  
axItdfLoad 179  
axItdfSave 180  
axITechnologyType 1010  
axITempDirectory 1397  
axITempFile 1398  
axITempFileRemove 1399

axITestPoint 1566  
axIText2Lines 1561  
axITextOrientationCopy 412  
axITransformObject 413  
axITriggerClear 1011  
axITriggerPrint 1012  
axITriggerSet 1013  
axUIAppMode 689  
axUICmdPopupSet 547  
axUIColorDialog 693  
axUIConfirm 694  
axUIConfirmEx 695  
axUIControl 697  
axUIDataBrowse 754  
axUIEditFile 736  
axUIGetUserData 1330  
axUIMenuChange 699  
axUIMenuDebug 701  
axUIMenuDelete 702  
axUIMenuDump 692  
axUIMenuFind 703  
axUIMenuInsert 706  
axUIMenuLoad 691  
axUIMenuRegister 709  
axUIMultipleChoice 738  
axUIPopupDefine 1331  
axUIPopupDump 711  
axUIPopupSet 1333  
axUIPrompt 712  
axUIViewFileCreate 722  
axUIViewFileReuse 724  
axUIViewFileScrollTo 739  
axUIWBeep 740  
axUIWBlock 735  
axUIWClose 729  
axUIWCloseAll 714  
axUIWDisableQuit 741  
axUIWExpose 728  
axUIWExposeByName 742  
axUIWHelpRegister 731  
axUIWIconify 715  
axUIWIslIconic 716  
axUIWIslWindow 717  
axUIWMove 718

axUIWPerm 743  
axUIWPrint 733  
axUIWRedraw 719  
axUIWRedraw 734  
axUIWSetHelpTag 745  
axUIWSetParent 746  
axUIWShow 747  
axUIWSize 720  
axUIWTimerAdd 748  
axUIWTimerRemove 751  
axUIWUpdate 752  
axUIYesNo 726  
axUIYesNoCancel 753  
axUnfixAll 1563  
axUnsetVariable 588  
axUnsetVariableFile 589  
axVersion 1469  
axVersionIdGet 1473  
axVersionIdPrint 1474  
axViaZLength 1283  
axVisibleDesign 235  
axVisibleGet 237  
axVisibleLayer 239  
axVisibleSet 240  
axVisibleUpdate 1341  
axWFMAAnyExported 1021  
axWidth2Impedance 1564  
axWindowBoxGet 378  
axWindowBoxSet 379  
axWindowFit 548  
axWriteDeviceFile 1691  
axWritePackageFile 1693  
axXSectionAssign 249  
axXSectionCopy 250  
axXSectionCreate 251  
axXSectionCreateStackup 256  
axXSectionDelete 257  
axXSectionDeleteStackup 258  
axXSectionGet 259  
axXSectionLayerFunctions 266  
axXSectionLayerTypes 267  
axXSectionModify 268  
axXSectionRemove 269  
axXSectionRename 271

## Allegro SKILL Reference

---

axIXSectionSet 272  
axlZoneAccess 497  
axlZoneCreate 1111  
axlZoneDelete 417  
axlZoneSet 418  
axlZoomBbox 549  
axlZoomCenter 550  
axlZoomControl 551  
axlZoomFit 553  
axlZoomInOut 554  
axlZoomPoints 555  
axlZoomToDbid 556  
axlZoomWorld 558  
bBoxAdd 1524  
Cadence Customer Support 1616  
Callback Procedure: formCallback 858  
copyDeep 1629  
DLL Programming 1612  
Examples 1616  
Field / Control 758  
Input/Output Data Primitives 1614  
isBoxp 1630  
lastelem 1631  
letStar 1632  
listnindex 1633  
movedown 1634  
moveup 1635  
parseFile 1636  
parseQuotedString 1638  
Performance Considerations 1616  
pprintln 1639  
Programming 757  
Programming Restrictions, Cautions and Hints 1615  
propNames 1640  
SKILL Programming 1611

## **Allegro SKILL Reference**

---

# Before You Start

---

## About This Manual

This manual is for designers and engineers who use the Allegro PCB Editor SKILL functions to customize existing Allegro 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 Allegro PCB Editor for the physical design of PCBs and analysis of reliability, testability, and restructurability.

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 Allegro 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 Allegro PCB Editor to design boards, you should be familiar with the Allegro PCB Editor environment.

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

- Start Allegro PCB Editor
- Navigate in the Allegro 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>&lt;name&gt;</i>	Argument of type “name” required.
<i>dbid</i>	Instance of an Allegro PCB Editor database object (means “database id”)
<i>figure</i>	Geometric Allegro PCB Editor database object—for example, line, shape, or symbol are all Allegro PCB Editor figures. This is not to be confused with the Allegro PCB Editor’s database object “figure”, a special graphic denoting DRC markers and drill holes. To ensure clear distinction in this manual, “Allegro PCB Editor figure” denotes this special object.
<i>l_bBox</i>	A list of the coordinates of a bounding box. Coordinate pairs are lower left and upper right. For example,

```
(list( 100:100 200:200 ))
```

## Allegro SKILL Reference

### Before You Start

---

<i>t_layer</i>	A pair of names separated by a slash “/” denoting the name of an Allegro 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 Allegro PCB Editor object types and their associated names:

**Table 4-1 Allegro PCB Editor Object Types and Names**

Allegro 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 `ax1Name` functions in [Chapter 4, “Selection and Find Functions”](#).

## Finding Information in This Manual

The following table summarizes the topics described in this manual.

For Information About . . .	Read . . .
AXL operation and the relation between Allegro PCB Editor database and AXL functions:	<a href="#"><u>Chapter 1, “Introduction to Allegro PCB Editor SKILL Functions”</u></a>
<ul style="list-style-type: none"><li>■ AXL functions</li><li>■ AXL initialization, environment</li><li>■ Starting and stopping AXL</li><li>■ Debugging AXL programs</li><li>■ <i>dbids</i> and object persistence</li><li>■ Selecting Allegro PCB Editor database objects</li><li>■ AXL–SKILL database object types</li></ul>	
The structure of Allegro PCB Editor AXL database objects, and how they are related to each other. Lists attributes of each object type.	<a href="#"><u>Chapter 2, “The Allegro PCB Editor Database User Model”</u></a>
<ul style="list-style-type: none"><li>■ Database rules</li><li>■ Attribute types</li><li>■ Figure (geometric) database types</li><li>■ Logical database types</li><li>■ Property database types</li><li>■ Full attribute listing for all Allegro PCB Editor database objects</li></ul>	

## Allegro SKILL Reference

### Before You Start

---

<b>For Information About . . .</b>	<b>Read . . .</b>
Path structures and AXL functions that add path objects and other figure objects to the Allegro PCB Editor database.	<a href="#"><u>Chapter 15, “Database Create Functions”</u></a>
<ul style="list-style-type: none"><li>■ Path create functions</li><li>■ Create shape and rectangle functions</li><li>■ Create functions for line, pin, symbol, text, and via</li></ul>	
Read/Write access to Allegro PCB Editor database parameter objects.	<a href="#"><u>Chapter 3, “Parameter Management Functions”</u></a>
The select set and AXL functions for managing the select set and selecting single and multiple database objects.	<a href="#"><u>Chapter 4, “Selection and Find Functions”</u></a>
<ul style="list-style-type: none"><li>■ Point selection</li><li>■ Box selection</li><li>■ Selection by name and object <i>dbid</i></li><li>■ Find filter management</li><li>■ Selection set management</li></ul>	
AXL functions that operate on database objects in the same way as interactive Allegro PCB Editor commands, including functions to:	<a href="#"><u>Chapter 5, “Interactive Edit Functions”</u></a>
<ul style="list-style-type: none"><li>■ Delete objects</li><li>■ Show objects</li></ul>	
Reading database objects:	<a href="#"><u>Chapter 6, “Database Read Functions”</u></a>
<ul style="list-style-type: none"><li>■ Opening an Allegro PCB Editor design</li><li>■ Accessing standalone branch figures, properties, pads, and text</li></ul>	

## Allegro SKILL Reference

### Before You Start

---

<b>For Information About . . .</b>	<b>Read . . .</b>
AXL functions for	<a href="#"><u>Chapter 7, “Allegro PCB Editor Interface Functions”</u></a>
■ 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	
AXL functions for	<a href="#"><u>Chapter 8, “Allegro PCB Editor Command Shell Functions”</u></a>
■ Setting Allegro PCB Editor shell variables	
■ Sending a command string to the Allegro PCB Editor shell.	
AXL functions for	<a href="#"><u>Chapter 10, “User Interface Functions”</u></a>
■ Prompting and getting confirmation from the user, displaying text files	
■ Displaying and printing ASCII files	
AXL forms and functions for	<a href="#"><u>Chapter 11, “Form Interface Functions”</u></a>
■ 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	<a href="#"><u>Chapter 12, “Simple Graphics Drawing Functions”</u></a>
Writing AXL functions for	<a href="#"><u>Chapter 13, “Message Handler Functions”</u></a>
■ Setting up for user messages	
■ Displaying messages to users	

## Allegro SKILL Reference

### Before You Start

---

<b>For Information About . . .</b>	<b>Read . . .</b>
AXL functions for	<u><a href="#">Chapter 14, “Design Control Functions”</a></u>

■ Opening a design

■ Compiling, running edit check, and saving the current (symbol) design

■ Getting the type of the active design

■ Setting the Allegro 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 Allegro 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 SKILL Reference

### Before You Start

---

<b>For Information About . . .</b>	<b>Read . . .</b>
AXL functions for	<u><a href="#">Chapter 16, “Database Group Functions”</a></u>
<ul style="list-style-type: none"><li>■ Creating and removing database groups.</li><li>■ Adding and removing database objects from database groups.</li></ul>	
AXL functions for	<u><a href="#">Chapter 17, “Database Attachment Functions”</a></u>
<ul style="list-style-type: none"><li>■ Creating, changing, and deleting database attachments</li><li>■ Checking whether an object is a database attachment</li><li>■ Getting the ids of all database attachments</li><li>■ Getting a database attachment</li></ul>	
AXL functions for	<u><a href="#">Chapter 18, “Database Transaction Functions”</a></u>
<ul style="list-style-type: none"><li>■ Improving the performance and program memory use while updating many etch or package symbols in batch mode</li><li>■ Marking the start of a database transaction and returning the mark to the calling function</li><li>■ Writing a mark in the database to allow future rollback or commitment</li><li>■ Committing and undoing a database transaction</li></ul>	

## Allegro SKILL Reference

### Before You Start

---

<b>For Information About . . .</b>	<b>Read . . .</b>
AXL functions for	
<ul style="list-style-type: none"><li>■ Getting and setting current DRC modes and values for design constraints and ECset members</li><li>■ Creating, deleting, and getting the dbid of an ECset</li><li>■ Checking the syntax of a given value against the allowed syntax for a given constraint</li><li>■ Batching and tuning DRC updates from constraint changes made using axlCNS&lt;xxx&gt; functions</li></ul>	<u><a href="#">Chapter 19, “Constraint Management Functions”</a></u>
AXL functions for	<u><a href="#">Chapter 20, “Command Control Functions”</a></u>
<ul style="list-style-type: none"><li>■ Registering and unregistering SKILL commands with the command interpreter</li><li>■ Getting and setting the controls for line lock, active layer</li><li>■ Defining popups</li><li>■ Getting user data</li></ul>	
Polygon operation functions, attributes, and use models.	<u><a href="#">Chapter 21, “Polygon Operation Functions”</a></u>
Getting Allegro PCB Editor file names, and opening and closing files	<u><a href="#">Chapter 22, “Allegro PCB Editor File Access Functions”</a></u>
AXL functions for	<u><a href="#">Chapter 23, “AXL-SKILL Data Extract Functions”</a></u>
<ul style="list-style-type: none"><li>■ SKILL access to the extract command</li><li>■ Selecting sets of database objects as members of a view and applying an AXL function to each</li></ul>	

## Allegro SKILL Reference

### Before You Start

---

For Information About . . .	Read . . .
AXL functions for	<a href="#">Chapter 24, “Utility Functions”</a>
■ Calculating an arc center given various data	
■ Converting quantities to various units	
Using math utility functions.	<a href="#">Chapter 25, “Math Utility Functions”</a>
Odds and ends.	<a href="#">Chapter 26, “Database Miscellaneous Functions”</a>
Using logic access functions.	<a href="#">Chapter 30, “Logic Access Functions”</a>

---

## Other Sources of Information

For more information about Allegro 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 Allegro PCB Editor documentation set for your workbench of PCB design tools:

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

## Allegro SKILL Reference

### Before You Start

---

<b>For Information About...</b>	<b>Read...</b>
Loading logical design data and converting third-party mechanical data, including loading data from Concept™, netlists, and board mechanical data.	<a href="#"><u>Allegro PCB Editor User Guide: Transferring Logic Design Data</u></a>
Setting up the design and specifying design rules and controls, including instructions for specifying properties and constraints.	<a href="#"><u>Allegro PCB Editor User Guide: Preparing the Layout</u></a>
Placing components using Allegro PCB Editor, including automatic and interactive placement.	<a href="#"><u>Allegro PCB Editor User Guide: Placing the Elements</u></a>
Routing using Allegro PCB Editor, including interactive and automatic routing.	<a href="#"><u>Allegro PCB Editor User Guide: Routing the Design</u></a>
Design output, including renaming reference designators, creating drill and silkscreen data, and generating penplots.	<a href="#"><u>Allegro PCB Editor User Guide: Preparing Manufacturing Data</u></a>
Optional Allegro 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.	<a href="#"><u>Converting Third Party Designs and Mechanical Data</u></a>
Allegro PCB Editor commands, listed alphabetically.	<a href="#"><u>Allegro PCB and Package Physical Layout Command Reference</u></a>
Allegro PCB Editor properties, extracts (examples), and reports.	<a href="#"><u>Allegro PCB Editor User Guide: Design Rules, Extract Data Dictionary, and Viewing Reports On Screen in HTML Format</u></a>
A comprehensive glossary for the Allegro PCB Editor user guides and reference manuals.	<a href="#"><u>Allegro PCB Editor User Guide Glossary</u></a>

## **Customer Support**

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

Cadence Online Support (COS) is available for customers who have a maintenance agreement with Cadence.

Here, you can also find technical information, including SKILL documentation and shareware code, online. COS provides the latest in Quarterly Incremental Releases (QIRs), case and product change release (CCR) information, technical documentation, solutions, software updates and more.

## **Allegro PCB Editor Users Mailing List Subscription**

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

### **To subscribe to the Allegro 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:

`<install_dir>/share/pcb/examples/skill`

## **User Discussion Forums**

You can find discussion groups for users of Cadence products at Cadence Community site.

---

# **Introduction to Allegro PCB Editor SKILL Functions**

---

## **Overview**

This chapter is a brief overview of the following:

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

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

## **AXL-SKILL in Allegro PCB Editor**

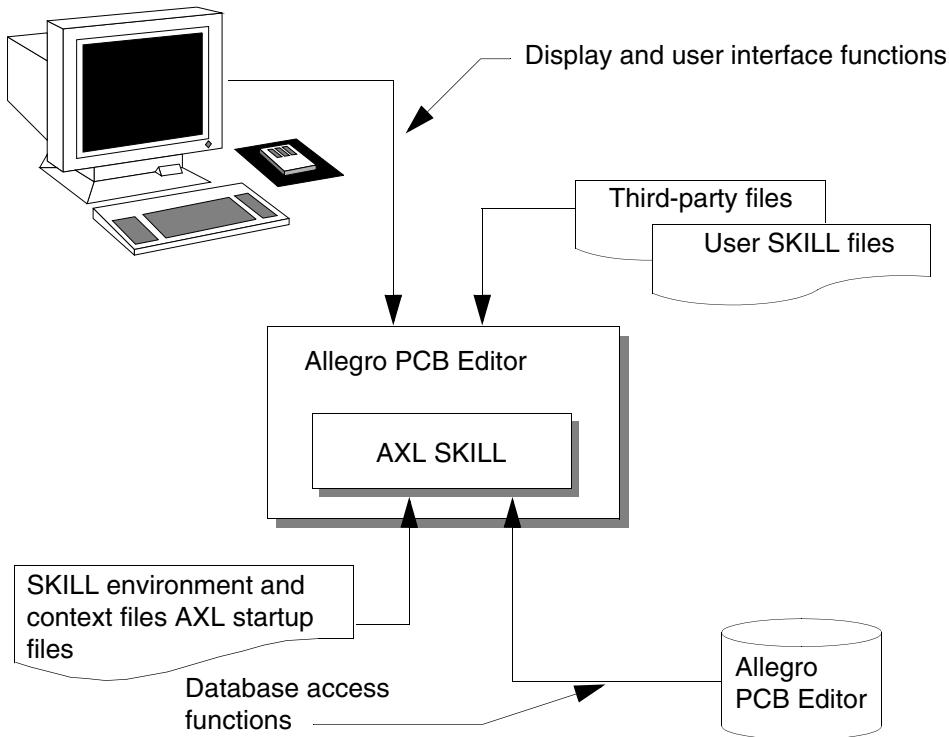
AXL-SKILL is a language processor contained in Allegro PCB Editor, as shown in the following figure.

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

## Allegro SKILL Reference

### Introduction to Allegro PCB Editor SKILL Functions

data using the core 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 Allegro PCB Editor command line.

AXL-SKILL initializes automatically when you start Allegro PCB Editor. As Allegro PCB Editor starts, it reads the Allegro 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 Allegro PCB Editor.

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

- For the locations specified below, Allegro 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 Allegro PCB Editor command line. The AXL-SKILL interpreter appears with the `skill>` prompt in place of the Allegro PCB Editor command line.

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

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

Type `exit` to close the AXL-SKILL interpreter and return to the Allegro 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 Allegro 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 Allegro PCB Editor graphic executables, such as `allegro_layout`, `allegro_engineer`, `allegro_prep`, `allegro_interactive`, and `allegro_layout`.

You can also program the Allegro 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 Allegro 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 Allegro 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 Allegro PCB Editor environment variable, `TELSKILL`.

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

## **AXL-SKILL Database**

Allegro 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 Allegro 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 Allegro PCB Editor from AXL.
- You run an Allegro 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 Allegro 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 Allegro 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 4, “Selection and Find Functions,”](#) for information about select set functions.

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

## Design Files

*Design files* are containers for Allegro 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 Allegro PCB Editor loads from a schematic or third-party netlist file:

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 Allegro PCB Editor connects or disconnects parts of the net. A completely connected net consists of one and only one branch.

## Layer Attributes

Each Allegro 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 Allegro PCB Editor class/subclass combination. A later section describes this structure in detail.

## Allegro PCB Editor Properties

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

## Allegro SKILL Reference

### Introduction to Allegro PCB Editor SKILL Functions

---

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 Allegro 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. (Allegro PCB Editor figure: <code>arc</code> <code>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 (Allegro PCB Editor figure: <code>line</code> <code>segment</code> ).
Path	Is a sequence of end-to-end lines and arcs on the same layer. Each segment can have a different width (Allegro 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 (Allegro PCB Editor figure: <code>unfilled</code> <code>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.
Via	Is a connecting drill path between layers with associated padstack.

## Allegro SKILL Reference

### Introduction to Allegro PCB Editor SKILL Functions

---

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 Allegro PCB Editor Colors with AXL-SKILL

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

#### Design Object

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

AXL API calls, including `axlLayerGet` (“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 = axlLayerGet("etch/top")
p->color->2
```

## **Allegro SKILL Reference**

### Introduction to Allegro PCB Editor SKILL Functions

---

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 Allegro 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

## **Allegro SKILL Reference**

### Introduction to Allegro PCB Editor SKILL Functions

---

- Text
- Polygons
- Shapes

#### **Types of Parameters**

Allegro PCB Editor parameters store feature or board level options to the Allegro 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**

<b>Object Name</b>	<b>Object Parts</b>
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 Allegro 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 88](#))
  - Branches ([Table 2-4 on page 88](#))
  - Design Files ([Table 2-5 on page 89](#))
  - DRCs ([Table 2-6 on page 90](#))
  - Lines ([Table 2-12 on page 93](#))
  - Paths ([Table 2-15 on page 100](#))
  - Polygons ([Table 2-17 on page 102](#))
  - Pins ([Table 2-16 on page 101](#))
  - Shapes ([Table 2-19 on page 103](#))
  - Symbols ([Table 2-20 on page 106](#))
  - Tees ([Table 2-22 on page 107](#))
  - Vias ([Table 2-24 on page 108](#))
  - Pads ([Table 2-13 on page 95](#))
  - Padstacks ([Table 2-14 on page 96](#))

- Symdefs ([Table 2-21](#) on page 107)
- Logical objects
  - Components ([Table 2-27](#) on page 110)
  - Functions ([Table 2-29](#) on page 111)
  - Function Pins ([Table 2-30](#) on page 112)
  - Nets ([Table 2-32](#) on page 113)
- Property dictionary objects ([Table 2-42](#) on page 121)
- Parameter objects
  - Design ([Table 2-45](#) on page 124)
  - Display ([Table 2-46](#) on page 125)
  - Layer Group ([Table 2-48](#) on page 126)
  - Layer ([Table 2-49](#) on page 127)
  - Textblock Group ([Table 2-50](#) on page 128)
  - Textblock ([Table 2-51](#) on page 128)

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

- Invoking the `axlShell` function or editing a new Allegro 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 Allegro PCB Editor database.

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

AXL does not update objects asynchronously.

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

- DRC objects are volatile.

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

You cannot *directly* change a DRC object.

The following Allegro 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 Allegro PCB Editor database treats etch figures differently from non-etch figures. The following are the etch figure rules. [Figure 2-1](#) on page 84 shows the connectivity model used by Allegro PCB Editor.

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

Allegro 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*.

## Allegro SKILL Reference

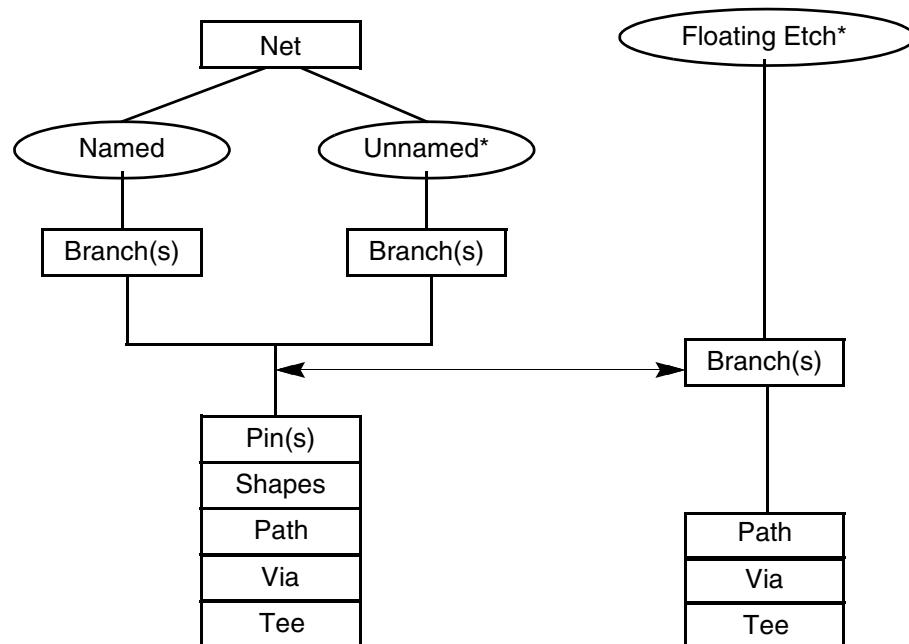
### The Allegro PCB Editor Database User Model

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

- Tees and branches are volatile.

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

**Figure 2-1 Allegro PCB Editor Connectivity Model**



<b>Tee</b>	Connection of three or more paths, added and removed as needed
<b>Via</b>	Can be deleted by ripup or glossing operations
<b>Path</b>	The net attribute on paths, vias and tees changes due to connectivity to pins and shapes.
<b>Line/Arc</b>	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	Allegro 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 Allegro PCB Editor Object Attributes

The following attributes are generic to all Allegro PCB Editor database objects. All Allegro 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 = can modify object directly
		<b>Note:</b> Except for parameter dbids all other objects are readOnly=nil. This means they cannot be modified via "dbid->object = value".

*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 Allegro PCB Editor, also share a common set of attributes. However, “figure” is not actually an attribute of any object. [Table 2-2](#) on page 87 lists the common figure attributes. In Allegro 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 Allegro PCB Editor object type. The Common Figure attributes (see [Table 2-2](#) on page 87) and the Generic Allegro PCB Editor Object attributes (see [Table 2-1](#) on page 86) 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 Single arc cline or line in Allegro**

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
startEnd	l_point	Start and end points of arc
width	float	Width of line
font	symbol/nil	Line font, etch and solid is nil.  Other types are 'HIDDEN' 'PHANTOM' 'DOTTED' 'CENTER'
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 SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-5 Design Attributes Represents the design root, axIDBGetDesign()**

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
designOutline	dbid	Design outline shape/rect, nil if none (layer BOARD GEOMETRY/DESIGN_OUTLINE)
diffpair	l_dbid	List of differential pairs
drcs	l_dbid	List of non-waived 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
keepinPlace	dbid	Place shape/rect keepin, nil if none (layer PACKAGE KEEPIN/ALL)
keepinRoute	dbid	Route shape/rect keepin, nil if none (layer ROUTE KEEPIN/ALL)
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
netclass	l_dbid	List of netclass constraints group
netgroup	l_dbid	List of netgroups in the design
objType	string	Type of object, in this case "design"
region	l_dbid	List of regions
padstacks	l_dbid	List of padstacks
pins	l_dbid	If a .dra, list of pins, else nil
symbols	l_dbid	List of symbol instances

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-5 Design Attributes, *continued*** Represents the design root, `axIDBGetDesign()`

Attribute Name	Type	Description
symdefs	l_dbid	List of symbol definitions
text	l_dbid	List of unattached text in design. These are not associated with any other object.
waived	l_dbid	List of waived DRCs
xnet	l_dbid	List of Xnets (no nets with VOLTAGE property)
zone	l_dbid	List of zones (only for Rigid-Flex designs)

**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)
fixed	t/nil	t = Allegro PCB Editor generated DRC nil = user defined DRC
name	string	Name of constraint that was violated
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	Domain of DRC, values can be: <ul style="list-style-type: none"> <li>■ "NET SPACING CONSTRAINTS"</li> <li>■ "PHYSICAL CONSTRAINTS"</li> <li>■ "DESIGN"</li> <li>■ "NET ELECTRICAL CONSTRAINTS"</li> <li>■ "SAME NET CONSTRAINTS"</li> <li>■ "EXTERNAL REFERENCE"</li> </ul>

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-6 DRC Attributes**

Attribute Name	Type	Description
violations	l_dbid	List of figures causing error (2 max)
waived	t/nil	t = waived DRC nil = regular
xy	point	Location of DRC marker

---

**Table 2-7 FUNCDEF Attributes: Function definition (logic element)  
Component definition has a list of funcdef (functions)**

Attribute Name	Type	Description
Also includes generic object attributes		
objType	string	"funcdef"
parent	dbid	compdef (Component Definition)
pins	l_dbid	list of funcdefPin (Function definition pins)
slot	string	Name of Slot (function def name)
type	string	Type of function

---

**Table 2-8 FUNCDEFPIN Attributes: Function definition pin (logic element)**

Attribute Name	Type	Description
Function definition has a list of these pins each with point has a link to the component def pin		
Also includes generic object attributes		
objType	string	"funcdefPin"
parent	dbid	pin (component pin definition)
name	string	Name of the pin
swapCode	integer	The swap code (pins with common codes can be swapped)
use	string	Use of pin (for example, UNSPEC)

---

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-9 FIGURE Attributes**

Attribute Name	Type	Description
Also includes generic and figure object attributes		
corners	string	Rnd Rect and Rect pad only: corners chamfered, dash separated:  <ul style="list-style-type: none"> <li>■ UR – upper right</li> <li>■ UL – upper left</li> <li>■ LR – lower right</li> <li>■ LL – lower left</li> <li>■ nil if pad type does not support</li> </ul>
drillChar	string	drill characters (max 3)
figureName	string	figure type
fill	t/nil	is figure filled (usually t)
inside	float	Donut pad only: inside diameter, 0 otherwise  <b>Note:</b> Outside diameter needs to be obtained from the bBox attribute (maxX - minX);
height	float	height of figure
objType	string	"figure"
radius	float	Rnd Rect and Rect pad only: corner radius, 0 otherwise
rotation	float	rotation absolute
width	float	width of figure
xy	point	Figure placement location

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-10 Group Attributes Groupings of Allegro Objects**

<b>Attribute Name</b>	<b>Type</b>	<b>Description</b>
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.
<b>Note:</b> This cannot be defined in SKILL. User-defined groups are considered "GENERIC."		

**Table 2-11 Module Attributes Module instance in design**

<b>Attribute Name</b>	<b>Type</b>	<b>Description</b>
Also includes generic object attributes		
bBox	bBox	Bounding box of all physical members of group
groupMembers	l_dbid	List of members of the module
name	string	Name of the module
objType	string	Type of object, in this case "group"
placeReplicate	t/nil	t if module is from Place Replicate, nil if a basic module
type	string	"MODULE"

**Table 2-12 Line Attributes Cline and line of Allegro objects**

<b>Attribute Name</b>	<b>Type</b>	<b>Description</b>
Also includes generic and figure object attributes		
isEtch	t/nil	t = a CLINE; nil = a LINE
lineType	s_type	a symbol: horizontal, vertical, odd
objType	string	Type of object, in this case "line"

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-12 Line Attributes, *continued* Cline and line of Allegro objects**

Attribute Name	Type	Description
parent	dbid	Path, polygon, or shape
startEnd	l_point	Start and end points
thermal	t/nil	Cline is a thermal relief.
width	float	Width of line
font	symbol/nil	Line font, etch and solid is nil.  Other types are 'HIDDEN 'PHANTOM 'DOTTED 'CENTER

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-13 Pad Attributes**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bBox	bBox	Bounding box. Coordinates are always relative.
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> nil denotes a null pad
figureName	string	Type of pad figure.  Examples are "CIRCLE", "SQUARE", "SHAPE", "NULL". May be nil if drill. Use axlPadFigureTypes() to obtain list of figures
flash	string	If of type FLASH, name of flash symbol otherwise an empty string ""  (Obsolete; use name attribute)
layer	string/ symbol	Pad layer  For regular layers and mask layers this is a string. For composite, internal, adjacent, backdrill layers this is a symbol
name	string	If type is a SHAPE or FLASH, name of symbol.
objType	string	Type of object, in this case "pad"
offset	l_point	Offset of pad (relative to pin/via origin)
parent	dbid	Padstack <i>dbid</i>
readOnly	t	Cannot be modified.
type	string	Pad type is one of: REGULAR, ANTI, THERMAL, or KEEPOUT
sides	int	Octagon pad only: Number of sides, 0 otherwise  Ranges between 6 and 64 as an even integer.
inside	float	Donut pad only: inside diameter, 0 otherwise  <b>Note:</b> Outside diameter needs to be obtained from the bBox attribute (maxX - minX);

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-13 Pad Attributes**

Attribute Name	Type	Description
radius	float	Rnd Rect and Rect pad only: corner radius, 0 otherwise
corners	string	Rnd Rect and Rect pad only: corners chamfered, dash separated:  <ul style="list-style-type: none"> <li>■ UR - upper right</li> <li>■ UL - upper left</li> <li>■ LR - lower right</li> <li>■ LL - lower left</li> </ul> nil if pad type does not support

**Table 2-14 Padstack (PPin/Via Definition) Attributes**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
definition	o_dbid	(see isPadRef) Points to padstack definition if this is padReference otherwise nil
derived	string	if a derived padstack name of original padstack, else nil
drillChar	string	drill characters (max 3)
drillDiameter	float	Drill hole diameter (0 if slot)
drillSizeWidth	float	width of slot, diameter if hole and extents of multidrill
drillSizeHeight	float	height of slot, diameter if hole and extents of multidrill
drillSize	string	Name of drill to use (optional)
drillOffset	point	offset of drill hole
drillFigureName	string	type of drill symbol (circle, square, and so on.)
drillFigureWidth	float	Width of drill symbol (for slots same as drillSizeWidth)

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-14 Padstack (PPin/Via Definition) Attributes, *continued***

Attribute Name	Type	Description
drillFigureHeight	float	Height of drill symbol (for slots same as drillSizeHeight)
drillNonStandard	string	Type of drill ( <code>nil</code> is standard) <i>(not supported by slots)</i>
drillToolSize	string	Description of actual drill. This field was introduced in 17.0 to describe the drill used. It should NOT be treated as a number.  Is optional and may be an empty string.
holeCounterAngle	integer	If counter sink reports angle of sink with range 0 to 90.
holeCounterDepth	float	If counter bore report depth of sink in user units.  This is maintained with an accuracy greater than the design
holeCounterDiameter	float	Diameter of counter bore/sink in user units
holeCounterTolerance	l_float	If counter bore/sink a list of two design units reporting the + and - tolerance of bore/sink
holeCounterType	string	If counter bore reports "bore"  If counter sink reports "sink"  else is <code>nil</code>
holeTolerance	l_float	A list of two deign units reporting the + and - drill hole tolerance. If a slot this is the 'X' or NC routing distance tolerance.
holeToleranceY	l_float	If slot is list of two design units reporting the + and - 'Y' drill hole tolerance. This is the routing drill tolerance.
multiDrillData	l_values	If not multidrill then <code>nil</code> otherwise list of:  (rows columns clearanceX clearanceY staggered)  Both clearances are in <code>dbreps</code> and staggered is t/ <code>nil</code>
holeType	string	Type of hole (circle_drill, oval_slot, and so on.)

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-14 Padstack (PPin/Via Definition) Attributes, *continued***

Attribute Name	Type	Description
keepout	nil	This is obsolete for 17.0.  Use routekeep layer for padstacks
isPadRef	t/nil	t = padstack is a padstack reference.  This means that the actual padstack is a template and the start and end layers of the padstack are dynamically mapped, depending upon its use with a pin or a via.
isThrough	t/nil	t = through padstack
name	string	Padstack name
objType	string	Type of object, in this case "padstack"
pads	l_dbid	List of pads
parent	dbid	Design
padSuppression	t/nil	Does padstack have Pad Suppression enabled. This is for the legacy artwork based pad suppression.  The dynamic pad suppression ignores this option.
plating	string	One of "Plated" or "Non-Plated".  In 17.0 "Plating-Optional" is obsolete.
pluralVia	t/nil	is padstack marked as a plural (multi-net) compatible padstack
prop	l_dbid	Padstack's properties
spanLockCount	integer	Reports non-zero if padstack will not expand, if new layers are added or deleted.  0 if padstack will expand (this is the default).
		In the future, this may report the number of layers the padstack covers in lock layer span mode.
startEnd	lt_layer	Start and end layer of padstack

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-14 Padstack (PPin/Via Definition) Attributes, *continued***

Attribute Name	Type	Description
type	string	Type of padstack; valid values are: <ul style="list-style-type: none"> <li>■ through</li> <li>■ smd</li> <li>■ bbvia</li> <li>■ uvia</li> </ul> <p><b>Note:</b> Kept for backward compatibility; you should migrate to usage field</p>
usage	usage	Usage of pad. Superset of type. Valid values are: <ul style="list-style-type: none"> <li>■ through</li> <li>■ smd</li> <li>■ bbvia</li> <li>■ uvia</li> <li>■ die_pad_legacy</li> <li>■ through_pin</li> <li>■ surface_pin</li> <li>■ through_via</li> <li>■ die_pad</li> <li>■ bond_finger</li> <li>■ fiducial</li> <li>■ slot</li> <li>■ mechanical_hole</li> <li>■ tooling_hole</li> <li>■ mounting_hole</li> </ul>
uvia	t/nil	A sub-type of bbvias, to differentiate in constraint system.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-14 Padstack (PPin/Via Definition) Attributes, *continued***

Attribute Name	Type	Description
Backdrill Data: These attributes are nil, until set		
backdrillDiameter	float	Diameter of backdrill
backdrillFigureName	string	type of drill symbol (circle, square, and so on.)
backdrillFigureChar	string	Backdrill characters (max of 3)
backdrillFigureHeight	float	Height of backdrill figure
backdrillFigureWidth	float	Width of backdrill figure

**Note:** axIDBGetPad can be utilized to more easily query pad layer data.

---

**Table 2-15 Path Attributes Path clines or lines in Allegro**

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 the 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	owner (Branch, symbol, shape)
segments	l_dbid	List of arc and line figures in this path
symbolEtch	dbid	Symbol owner if etch.
startEnd	l_t_layer	If bond-wire start and end layers; nil if not bondwire.  If bondwire has a disconnected end, then the start and/or end will be nil

**Table 2-16 Pin Attributes Pin (board and symbol editors) (generic and figure elements)**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bondpad	class/ subclass	name of pad if pin is a bond pad (for example, "ETCH/BOND_TOP"), otherwise nil
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
isMirrored	t/nil	t = pin is mirrored
isThrough	t/nil	t = pin is a throughhole
mirrorType	string	Type of mirror.
name	string	Padstack name of this pin nil if unplaced component pin
number	string	Pin number
objType	string	Type of object, in this case "pin"
pads	l_dbid	Unordered list of pads. <sup>1</sup> To access a particular pad, use axlDBGetPad().

**Table 2-16 Pin Attributes, *continued*** **Pin (board and symbol editors) (generic and figure elements)**

Attribute Name	Type	Description
parent	dbid	<i>dbid</i> of symbol owning this pin <sup>2</sup>  <i>nil</i> 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 <sup>2</sup>
testPoint	t_layer/nil	<i>t_layer</i> , denotes layer of testpoint <i>nil</i> = pin is not a testpoint
use	string	Pin use as shown by show element
xy	point	Location of pin in absolute coordinates

May be *nil* if component is unplaced.  
Will say etch/(unknown) if unplaced component.

**Note:** Pins 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.

**Table 2-17 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

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-17 Polygon Attributes, *continued***

Attribute Name	Type	Description
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 2-18 Rat\_T Attributes Rat\_T (figure element)**

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 2-19 Shape Attributes Shapes or rectangles in Allegro**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
autoKeepoutType	string	If shape is a route keepout and derived from a padstack. This indicates type (adjacent_rko, backdrill_rko or same_layer_rko)
autoObject	dbid	If this a an auto-generated route keepout this shows the pin or via that generated it.
bBox	bBox	Bounding box

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-19 Shape Attributes, *continued*** Shapes or rectangles in Allegro

Attribute Name	Type	Description
cavity	t/nil	If a boundary shape is for cavity generation (embedded design). Cavity shapes are generated automatically based on the rki.
children	l_dbid	Used when a Boundary Shape points to a list of dynamic shapes
fill	g_fill	<p><code>t = filled; nil = unfilled</code></p> <p><code>(axlFillType axlFillType) - xhatch</code></p> <p>Each <code>axlFillType</code> has spacing width, origin, and angle.</p>
fillet	t/nil	shape is a fillet (teardrop)
fillOOD	t/nil	<p>Dynamic fill is out of date.</p> <p><code>t = shape needs fill updating</code>  (Only dynamic shapes can be t)</p> <p><code>nil = shape does not refill</code></p>
voids	list	List of o_polygons
isHole	t/nil	<code>t = polygon is a hole</code>
isRect	t/nil	<code>t = polygon is a rectangle</code>
nSegs	integer	Number of segments in polygon
objType	string	Type of object, in this case "shape"
parent	dbid	Branch (if etch shape), Symbol, shape (for voids) <code>nil = no parent</code>

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-19 Shape Attributes, *continued*** Shapes or rectangles in Allegro

Attribute Name	Type	Description
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.
region	l_dbid/nil	Region owner if a region shape.
segments	l_dbid	Path boundary of shape
shapeAuto	l_dbid/nil	If dynamic shape list of generated shapes on the matching auto-gen ETCH or CAVITY layer
shapeBoundary	dbid/nil	If this shape is generated from a dynamic shape, this points to that shape.  Boundary shapes may either be used for ETCH or CAVITY (see state of cavity attribute)
shapelsBoundary	t/nil	This shape is a dynamic shape, for example, on BOUNDARY class.
taper	t/nil	shape is used for tapering
dynamicGroup	t/nil	If this a dynamic shape (BOUNDARY class) return its dynamic group object. This is where voiding instance overrides are stored.
symbolEtch	dbid	Symbol owner, if etch
vertices	list	outer boundary available as a list containing a point (vertex of a polygon) and a floating point number (radius of the edge from the previous to the present vertex).

---

**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 ax1SetFindFilter to set the find filter to auto-select the boundary shape when the user selects one of the auto-generated children.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-20 Symbol Instance Attributes symbol**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
children	l_dbid	attached objects. This is non-etch data. Pins and pin escapes are pins and etchChildren respectively.
component	dbid	Component instance associated with symbol (nil if no component)
definition	dbid	Symbol definition
isMirrored	t/nil	t = symbol is mirrored
mirrorType	string	specific mirror type in use (YES, NO or GEOMETRY (APD+ only)).
name	string	Symbol name
objType	string	Type of object, in this case "symbol"
parent	dbid	Design (no other parent possible for symbols)
pins	l_dbid	List of pins
refdes	string/nil	Reference designator nil if no associated component (for example, mechanical)
rotation	float	Symbol rotation
type	string	Symbol type is one of: PACKAGE, MECHANICAL, FORMAT, SHAPE, FLASH or DRAFTING
xy	point	Symbol location
embedded	t/nil	symbol placed on embedded layer
embeddedLayer	string	layer of placed symbol or nil if external
embeddedMethod	string	method (CHIP_UP or CHIP_DOWN)
embeddedAttach	string	attachment method (DIRECT_ATTACH or INDIRECT_ATTACH)
etchChildren	l_dbid	does symbol have attached etch (typically pin escapes or shapes)

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-20 Symbol Instance Attributes, *continued symbol***

Attribute Name	Type	Description
fixedByTestPoint	t/nil	OBSOLETE - kept for backwards compatibility. Use axlDBIsFixed(<dbid>) or axlDBControl(?testPointFixed) instead.
layer	string	Placement layer (applies to zones and surface placement)
zone	dbid	placement zone (nil if no zones). Zone name is zoneDbid->name

**Table 2-21 Symdef (Symbol Definition) Attributes**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bBox	bBox	bounding box
		Because a symdef s not placed the center point of the box is 0,0
children	l_dbid	List of children making up symbol
instances	l_dbid	List of symbol instances using this symbol
name	string	Name of symbol definition
objType	string	Type of object, in this case "symdef"
parent	dbid	Design root
pins	l_dbid	List of pins
type	string	Symbol type is one of the following: PACKAGE, MECHANICAL, FORMAT, SHAPE, or FLASH

**Table 2-22 Tee Attributes**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid	Branch owner

**Allegro SKILL Reference**  
 The Allegro PCB Editor Database User Model

---

**Table 2-22 Tee Attributes, *continued***

Attribute Name	Type	Description
objType	string	Type of object, in this case "tee"
parent	dbid	Branch
readOnly	t	User cannot directly modify
xy	point	Location

**Table 2-23 Text Attributes Text in Allegro**

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 2-24 Via Attributes Via (board and symbol editors) (figure element)**

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid/nil	Branch owner
definition	dbid	Padstack definition
fixedByTestPoint	t/nil	OBSOLETE- kept for backwards compatibility. Use axlDBIsFixed(<dbid>) or axlDBControl(?testPointFixed) instead.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-24 Via Attributes, *continued* Via (board and symbol editors) (figure element)**

Attribute Name	Type	Description
isMirrored	t/nil	t = via mirrored
isThrough	t/nil	t = via is a through hole padstack
mirrorType	string	specific mirror type in use ("YES" "NO" "GEOMETRY"). Geometry indicates via is mirrored on same layer. Applies only APD+.
net	dbid/nil	net dbid
relations	list	List of relations of via Currently supports return path capability Format is one of the following: nil ('ground l_dbidVia) ('signal l_dbidVia)
name	string	"VIA"
objType	string	Type of object, in this case "via"
pads	l_dbid	Unordered list of pads. To access a specific pad, use ax1DBGetPad.
pluralVia	t/nil	is via a plural via (Multi-net)
pluralViaConnect	t/nil	is via a plural connect via (Multi-net)
parent	dbid	Typically the branch
rotation	float	Via rotation
symbolEtch	dbid/nil	If part of a symbol the symbol dbid
testPoint	t_layer/nil	t_layer denotes layer of testpoint
xy	point	Location of via

## Logical Database Types

Allegro PCB Editor logical database objects have these generic attributes (see [Description of Database Objects](#) on page 82): objType, prop, and readOnly. Logical database objects do not have other attributes in common. The tables below list the attributes for each Allegro PCB Editor logical type.

**Table 2-25 Bus Attributes (logic element)**

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”
lock	t/nil	Is membership locked for editing (vector buses)

**Table 2-26 Compdef Attributes Component Definitions (logic element)**

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 2-27 Component Attributes Component instances (logic element)**

Attribute Name	Type	Description
Also includes generic object attributes		

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-27 Component Attributes, *continued*** **Component instances (logic element)**

Attribute Name	Type	Description
class	string	Component classification
compdef	dbid	dbid of component definition (COMPDEF)
deviceType	string	Device type of component (see COMPDEF)
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 2-28 Diffpair Attributes (logic element)**

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

**Table 2-29 Function Attributes**

Attribute Name	Type	Description
Also includes generic object attributes		
name	string	Function designator

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-29 Function Attributes, *continued***

Attribute Name	Type	Description
objType	string	Type of object, in this case "function"
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 2-30 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 2-31 MATCH\_GROUP Attributes (logic element)**

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

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-31 MATCH\_GROUP Attributes (logic element)**

Attribute Name	Type	Description
type	string	"MATCH_GROUP"

**Note:** For more information, see `axlMatchGroupCreate`.

**Table 2-32 Net Attributes Net Definitions (logic element)**

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)
<b>Note:</b> Island shapes causes the count to be not one, even if all pins are connected.		
objType	string	Type of object, in this case "net"
pinpair	l_dbid	List of pinpairs associated with net (1)
<b>Note:</b> If a net is a member of an xnet, all pinpairs appear on the xnet.		
ratsnest	l_dbid	List of ratsnest for net.
ratsnest On	t/nil	State of ratsnest display for the 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 ( <code>mg_dbid</code> , <code>t_relatePropDelay</code> ).
For more information, see <code>axlMatchGroupCreate</code>		
isBundled	t/nil	t = net contains at least one bundled ratsnest.
scheduleLocked	t/nil	t = net schedule cannot be changed
unconnected	integer	Number of remaining connections. This does not include connections to unplaced symbols.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-32 Net Attributes, *continued*** Net Definitions (logic element)

Attribute Name	Type	Description
unplaced	integer	Number of unplaced pins.

**Note:** If net is a member of an xnet then all pinpairs will appear on the xnet.

For more information on the rpd attribute see `axlMatchGroupCreate`.

**Table 2-33 NETCLASS Attributes: NetClass objects for constraint grouping**

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of nets, xnets, buses or differential pairs.
name	string	Name of the net class
objType	string	"group"
type	string	"NETCLASS"
electrical	t/nil	If part of the electrical and same net domain
physical	t/nil	If part of the physical and same net domain
spacing	t/nil	If part of the spacing and same net domain

**Note:** A NetClass can be a member of one or more domains but its name must be unique across all domains.

**Note:** Constraint overrides can be added to netclass via properties.

**Table 2-34 REGION Attribute: Region objects for constraint grouping**

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	List of constraint area shapes.
name	string	Name of the region
objType	string	"group"
type	string	"REGION"

**Table 2-35 CLASS Table Attribute**

Attribute Name	Type	Description
name	string	Name of ecset
netclass1	dbid	net class entry
netclass2	dbid	second net class entry (may be nil)
region	dbid	region class for table entry (may be nil)
physical	dbid	physical cset associated with entry or nil
spacing	string	spacing cset associated with entry or nil
sameNet	string	sameNet cset associated with entry or nil
assembly	string	physical assembly cset associated with entry (APD+ only)
objType	string	"classTable"
readOnly	t/nil	Cannot be modified
prop	l_dbid	List of properties on table entry. Typically where constraint overrides on entry exist. Also duplicates the cset names.

**Note:** These are the class-class, class-class-region and class-region constraint table entries. See `axlCnsClassTableCreate`.

**Table 2-36 Pinpair Attributes** (logic element)

Attribute Name	Type	Description
Also includes generic object attributes		
ecsetDerived	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"
parent	l_dbid	Net or xnet owning the pinpair.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-36 Pinpair Attributes, *continued*(logic element)**

Attribute Name	Type	Description
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 2-37 NET\_GROUP Attributes A net group**

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	members of the group (net_groups, nets, xnets, diff pairs, buses)
name	string	Name of group
objType	string	"group"
type	string	"NET_GROUP"
isInterfaceTop	t/nil	If t, group is the top of a definition-driven interface instance.

**Note:** Use `axlDBCreateGroup` family of commands to add, delete and modify these groups.

**Table 2-38 PORT\_GROUP Attributes Groupings of Allegro pin objects**

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	members of the group (port_groups, pins, function pins)
name	string	Name of group

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-38 PORT\_GROUP Attributes, continued** Groupings of Allegro pin objects

Attribute Name	Type	Description
objType	string	"group"
type	string	"PORT_GROUP"
isInterfaceTop	t/nil	If t, group is the top of a definition-driven interface instance.

**Note:** PORT\_GROUPS are groupings of Allegro pin objects.

**Table 2-39 ZONE Attributes Groupings of Allegro zone groups**

Attribute Name	Type	Description
Also includes generic object attributes		
groupMembers	l_dbid	members of the group (shapes for region, keepout rooms). may have additional members in the future.
name	string	Name of zone
objType	string	"group"
type	string	"ZONE_GROUP"
region	string	name of region associated with zone (may be nil)
room	string	name of room associated with zone (may be nil)
shapeBoundary	l_dbid	boundary of zone (PRIMARY group is typically nil) Only one boundary dbid is supported.
stackup	string	name of stackup associated with zone (see axIXSectionGet)

**Table 2-40 RATSNEST Attributes** Ratsnest definitions (logic element)

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 the same branch)

**Table 2-40 RATSNEST Attributes, *continued*** Ratsnest definitions (logic element)

Attribute Name	Type	Description
pins	l_dbid	The two pins (or ratTs) that the rats connect
pwrAndGnd	t/nil	Net is power and ground scheduled
ratnest	l_dbid	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 2-41 XNET Attributes** eXtended Net (logic element)

Attribute Name	Type	Description
Also includes generic object attributes		
bus	dbid	Bus dbid if part of a bus.
diffpair	dbid	differential pair dbid if part of a differential pair.
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 that has this xnet as a member, lists as a list of lists ( <code>mg_dbid t_relatePropDelay</code> ).
type	string	“XNET”.

**Note:** This attribute can only be defined 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 Allegro 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 Allegro 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 Allegro 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 2-42 Property Dictionary Attributes**

Attribute Name	Type	Description
Also includes generic object attributes		
dataType	string	Data type for property value (see <a href="#">Allowed Property Data Types</a> on page 120)
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.
		<b>Note:</b> This attribute is valid only for user-defined properties. For pre-defined Allegro PCB Editor properties, the useCount value will always be 0.
write	t/nil	t = writable or user defined nil = read-only or Allegro PCB Editor pre-defined

## Parameter Database Types

The following Allegro 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 `axlSetParam`, respectively. See [Chapter 3, “Parameter Management Functions”](#) for functions that provide easier access to layers and other parameter objects.

## Allegro SKILL Reference

### The Allegro PCB Editor Database User Model

---

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

**Table 2-43 Artwork Parameter Attributes**

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

**Table 2-44 Artwork Film Parameter Attributes**

Attribute Name	Type	Description
Includes no generic object attributes		
domains	I_string	List of domains where film is visible. Supported values are ipc2581, pdf, artwork, and visibility.
drawHolesOnly	t/nil	For pins and vias draw only the holes.
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 <code>art_aper.txt</code> . 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	I_string	List of layers comprising film
ipc2581	I_string	List of IPC2581 Layer mappings. Indicates where film should be used in IPC25281 output. Supported values are inner, outer, misc, doc, and soldermask.
mirrored	t/nil	Specifies whether to mirror the photoplot output.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-44 Artwork Film Parameter Attributes, *continued***

Attribute Name	Type	Description
name	string	Name of film
negative	t/nil	Is film positive (nil) or negative (t)
objType	string	Type of object, in this case "artwork"
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.
sequence	integer	Sequence number for PDF ordering.  Range is 0 to 255. If two films share the same number, the database ordering determines ties.
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.
vectorBasedPad	string	Extracts information about vector based pad behavior from the artwork control dialog box.

**Table 2-45 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 Allegro 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 SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-46 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 2-47 ECset Parameter Attributes Electrical Constraint Set (parameter element)**

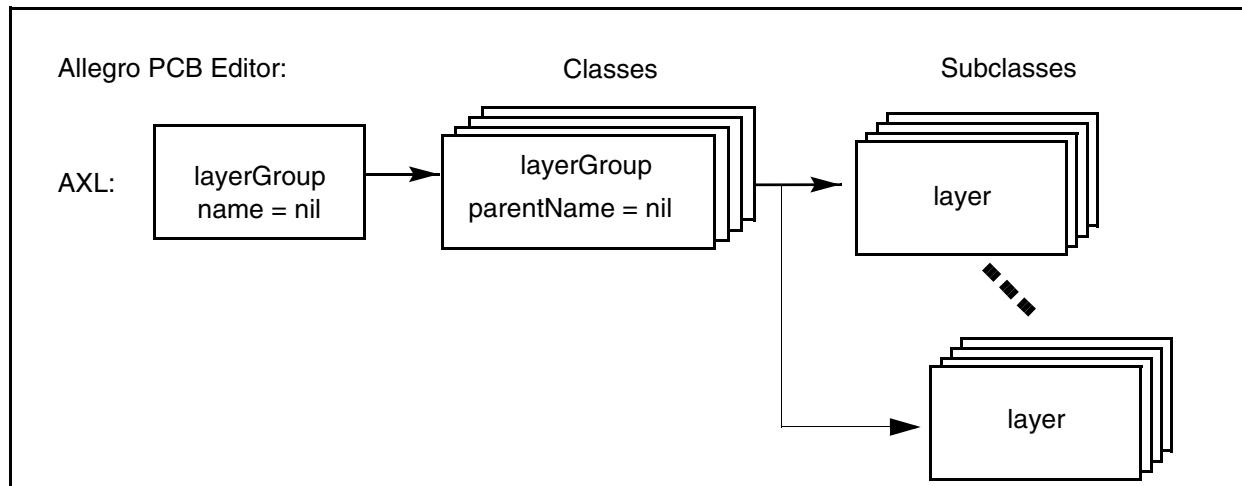
Attribute Name	Set?	Type	Description
Includes generic object attributes			
locked		t/nil	Cset locked from UI based editing
members		l_dbid	List of electrical constraints in set
name		string	Name of ECset
objType		string	Type of object, in this case "ecset"
prop		l_dbid	List of user-defined properties 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.

---

# **Allegro SKILL Reference**

## The Allegro PCB Editor Database User Model

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



**Table 2-48 Layer Group Parameter Attributes (Allegro Classes)**

<b>Attribute Name</b>	<b>Set?</b>	<b>Type</b>	<b>Description</b>
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	List of subclasses belonging to this class
isEtch	no	t/nil	Is an etch layer
name		string	Name of this class
nChildren	no	integer	Number of subclasses in class
objType	no	string	Type of object, in this case "paramLayerGroup"
pattern	yes	integer	Pattern for layer; -1 if all child layers not the same pattern (see <a href="#">Layer Parameter Attributes (Allegro Subclasses)</a> table on page 127)
visible	yes	t/nil	All subclasses of this class are visible

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-49 Layer Parameter Attributes (Allegro Subclasses)**

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
color	yes	integer	Subclass color number
drcPhotoType	no	string	"Positive" or "negative" (applies only to etch)
isEtch	no	t/nil	Is an etch layer
material	NO	string	Layer material (etch only)
name	no	string	Name of this subclass
number	no	integer	Layer number, while this is reported for both etch and non-etch layers it is only meaningful for etch layers where it shows the stackup order.
nextLayer	no	string	Name of next layer in stackup (etch only)
objType	no	string	Type of object, in this case "paramLayer"
parentname	no	string	Name of owning class
pattern	yes	integer	Pattern of layer where 0 is default solid pattern. For maximum patterns, see <a href="#">axlColorGet('pattern')</a> . Pattern style for each pattern number can be found in the color192 dialog.
thickness	no	string	Layer thickness (etch only)
type	no	string	"Positive" or Negative (etch only)
userDefined	No	t/nil	Is layer user defined. User defined layers could have been added by the user.
			Note for etch layers — all are user-defined, but you typically cannot delete or rename the TOP or BOTTOM.
visible	yes	t/nil	All subclasses of this class are visible

**Note:** The Allegro PCB Editor class/subclass system is modeled in AXL as layers. The previous drawing figure shows how AXL layers are mapped to Allegro PCB Editor class/subclasses. The define etch form in Allegro PCB Editor does not match this model. AXL does not support access to the dielectric pseudo-layers nor does it support analysis layer attributes

---

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

such as material and thickness. To access these records, use Allegro PCB Editor's technology file feature.

**Table 2-50 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"

**Table 2-51 Textblock Parameter Attributes**

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
charSpace	Yes	float	Spacing between characters in design units
height	Yes	float	Character height in design units (includes descender)
lineSpace	Yes	float	character line to line space in design units
name	no	string	Character block number
objType	no	string	Type of object, in this case "paramTextBlock"
photoWidth	Yes	float	character line width in design units
width	Yes	float	character width in design units

**Table 2-52 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.

**Allegro SKILL Reference**  
The Allegro PCB Editor Database User Model

---

**Table 2-52 Testprep Parameter Attributes, *continued***

Attribute Name	Set?	Type	Description
bareBoard	yes	t/nil	If nil, you can only test component pins on non-component 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.
executeInc	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.
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.

**Table 2-52 Testprep Parameter Attributes, continued**

Attribute Name	Set?	Type	Description
testPadDB	no	dbid	<i>dbid</i> of the testPad
testPadType	yes	symbol	Type of padstacks for probes: <code>smd</code> , <code>through</code> , or <code>either</code> .
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 <code>PSMPATH</code> 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 <code>t</code> , use Alphabetic extension, <code>nil</code> use numeric extension.
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.

---

## 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 Allegro 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.

## **axlcreate – Obsolete**

axlcreate

This interface is obsolete. It is kept to support existing SKILL code.

Use [axlFilmCreate](#).

## **axlDBGetTextBlockCount**

```
axlDBGetTextBlockCount()  
    )  
=> x_textBlockCount
```

### **Description**

Returns a count of the number of text blocks defined.

### **Arguments**

NA

### **Value Returns**

*x\_textBlockCount* A count of the number of text blocks defined.

### **Examples**

```
numTextBlocks = axlDBGetTextBlockCount()  
printf("This database has %d text blocks\n" numTextBlocks)
```

## axIDBGridGet

```
axIDBGridGet(  
    nil)  
    ==> lt_grids  
  
axIDBGridGet(  
    t_gridName)  
    ==> og_grid
```

### Description

This command returns current grid values. Function has two modes:

- if grid name is nil returns list of names
- If given a grid name return its grid characteristics (see below)

**Note:** Reserved grid name is "non-etch" otherwise grid names follow Allegro ETCH subclass names.

Use [axIDBDisplayControl](#) to control grid color and visibility.

Grids have the following attributes:

Name	Type	Description
objType	string	Name of the object - grids
readOnly	nil	can modify object
name	string	Name of grid
xOrigin	dbrep	X origin of grid
yOrigin	dbrep	Y origin of grid
xMajor	dbrep	Major X spacing of grid (read-only)
yMajor	dbrep	Major Y spacing of grid (read-only)
xGrids	I_dbrep	Spacings X of grid (always a list of dbreps)
yGrids	I_dbrep	Spacings Y of grid (always a list of dbreps)

## Arguments

*t\_gridName* name of grid or `nil` to get all grid names

## Value Returned

- `lt_grids` – list of grids
- `og_grid` – disembodied property list containing grid settings

## See Also

[ax1DBGridSet](#)

## Example

Run the following code to get all grids and print them.

```
grids = ax1DBGridGet(nil)  
  
foreach(g grids  
    grd = ax1DBGridGet(g)  
    printf("GRID name=%s  values=%L\n", grd->name, grd))
```

## axIDBGridSet

```
ax1DBGridSet(og_grid)==>t/nil
```

## Description

This command modifies the grid settings in the design.

In addition to the grid names (see `axIDBGridGet`), two symbolic grid names are available:

- 'all – sets all grid values
  - 'etch – sets all ETCH grid values

As a convenience when setting a single the xGrids or yGrids attribute, you can use a float.

Both xMajor and yMajor values are automatically determined by the sum of the spacings in xGrids and yGrids respectively.

## Notes:

- ❑ Non etch grids may not have multiple spacings. We only use the first grid seen.
  - ❑ Setting grids is not undo-able (this may change in the future).
  - ❑ Etch grids names are the same as ETCH layer names. This may change in the future.
  - ❑ Origin values must be within drawing extents or 0.
  - ❑ If Grid dialog is open it will not be updated when you change the grid settings using this API command.

## Arguments

*og\_grid* a grid disembodied property list from [axlDBGGridGet](#)

## Value Returned

`t` if the command is successful and the grid is changed, `nil` in case of failure.

## See Also

[axlDBGridGet](#), [axlDBDisplayControl](#)

## Examples

### 1. Modify TOP grid settings

```
grid = axlDBGridGet("TOP")
grid = axlDBG
```

### 2. Modify all grids (note allow xGrids and yGrids to NOT be list)

```
grid = axlDBGridGet("TOP")
grid->name = 'all
grid->xGrids = 5.0
grid->yGrids = 5.0
axlDBGridSet(grid)
```

### 3. Modify all etch grids

```
grid = axlDBGridGet("TOP")
grid->name = 'etch
grid->xGrids = '(5.0 7.0)
grid->yGrids = '(5.0 6.0)
axlDBGridSet(grid)
```

## **axlDBTextBlockCreate**

```
axlDBTextBlockCreate(  
    x_blockTemplate  
    ?width f_width  
    ?height f_height  
    ?lineSpace f_lineSpace  
    ?charSpace f_charSpace  
    ?photoWidth f_photoWidth  
) => x_textBlock/nil
```

### **Description**

Creates a new text block from the template block number provided. By providing optional text block characteristics, you can get available text blocks by:

```
lst = axlGetParam("paramTextBlock")
```

### **Arguments**

<i>x_blockTemplate</i>	Text block number (an integer) to use as template. 0 (zero) can be used to start with a new empty text block
<i>f_width</i>	Width of an individual character in design units
<i>f_height</i>	Height of an individual character in design units
<i>f_lineSpace</i>	Spacing between lines of text in design units
<i>f_charSpace</i>	Spacing between consecutive characters in design units
<i>f_photoWidth</i>	Width in design units of the strokes used to draw characters

### **Value Returned**

<i>x_textBlock</i>	New text block
<i>nil</i>	Returned if the command fails. Typically, this happens when you have exhausted the number block Allegro provides, or one of the parameters is not of the correct data type.



## See Also

[axlGetParam](#), [axlSetParam](#), [axlDBTextBlockCompact](#)

## Examples

- Create a new text block based upon text block 1 but change width and height  
`blockNum = axlDBTextBlockCreate(1 ?width 15.0 ?height 16.0)`
- Create a new text block without using a template. All parameters must be specified  
`block2 = axlDBTextBlockCreate(0 ?width 25.0 ?height 35.0 ?lineSpace 40.0  
?photoWidth 5.0 ?charSpace 4.0)`

## **axlDBTextBlockFindName**

```
axlDBTextBlockFindName (
    t_textBlockName
) => x_textBlockNumber/nil
```

### **Description**

Finds a text block based on its name.

### **Arguments**

*t\_textBlockName*      The name of the text block to be found.

### **Value Returned**

*x\_textBlockNumber*    Number of the text block found.

nil                          No text block found with the given name.

### **Examples**

- Find the text block with the name "Refdes".

```
textBlockNumber = axlDBTextBlockFindName ("Refdes")
if(textBlockNumber then
    printf("Text block Refdes is number %d\n" textBlockNumber)
else
    printf("There is no text block named Refdes\n")
```

## **axlDBTextBlockGetName**

```
axlDBTextBlockGetName(  
    x_textBlockNumber  
)=> t_name/nil
```

### **Description**

Returns the name associated with the given text block. Same as attribute `userName` in `axlGetParam( "paramTextBlock:<number>" )`.

### **Arguments**

`x_textBlockNumber` The index of the text block whose name it to be returned. Text blocks use a 1-based indexing.

### **Value Returns**

<code>t_name</code>	The text block name
<code>nil</code>	This text block has no name

### **Examples**

```
textBlockName = axlDBTextBlockGetName(textBlockNumber)
```

## **ax1DBTextBlockSetName**

```
ax1DBTextBlockSetName (
    x_textBlockNumber
    t_name
)
=> t/nil
```

### **Description**

Defines a name for a given text block.

### **Arguments**

<i>x_textBlockNumber</i>	The index of the text block whose name is to be defined. Text blocks use a 1 based indexing.
<i>t_name</i>	The name being defined for the text block. A <i>nil</i> indicates that there is no name.

### **Value Returns**

<i>t</i>	Success
<i>nil</i>	Error

### **Examples**

Define a name of "Refdes" for text block #3.

```
ax1DBTextBlockSetName(3 "Refdes")
```

## **axlExportXmlDBRecords**

```
axlExportXmlDBRecords(  
    t_fileName  
    lt_parmGroups/nil  
) -> t/nil  
  
axlExportXmlDBRecords(  
    nil  
) -> lt_parmGroups
```

### **Description**

This exports an Allegro Parameter file from the current design. It offers the same capability as (*File – Import – Parameter*). Side effect is creation of a `param_write.log` file.

### **Arguments**

<code>t_fileName</code>	Name of parameter file. Default extension is <code>.prm</code> and if not given a path component will locate the file via <code>PARAMPATH</code> . If filename is <code>nil</code> report back as a list the supported parameter groups.
<code>lt_parmGroups</code>	List of parameter groups to export or <code>nil</code> to export all.

### **Value Returned**

`t` if command is successfully executed, `nil` in case of an error

### **See Also**

[axlImportXmlDBRecords](#)

### **Examples**

1. In an existing dump, save all its settings and load into a new design

```
axlExportXmlDBRecords ("myparam" nil)  
axlOpenDesign (?design "newDesign")  
axlImportXmlDBRecords ("myparam")
```

2. Dump current parameter groups

```
axlExportXmlDBRecords (nil)
```

## axlFilmCreate

```
axlFilmCreate(
    t_filmname
    ?negative          t/nil
    ?undefineLineWidth f_width
    ?sequence          x_number
    ?rotation          x_angle
    ?xOffset           f_x
    ?yOffset           f_y
    ?shapeBoundingBox  f_value
    ?mirrored          t/nil
    ?fullContact       t/nil
    ?suppressUnconnectPads t/nil
    ?drawMissingPadApertures t/nil
    ?useApertureRotation t/nil
    ?suppressShapeFill   t/nil
    ?vectorBasedPad     t/nil
    ?drawHolesOnly      t/nil
    ?layers             lt_layers
    ?domains            lt_domains
    ?ipc2581            lt_ipcDomains
) -> t/nil
```

### Description

Creates a new artwork film or replaces an existing artwork film.

The terminology used matches the artwork dialog box. For more information on how each field is used, see the dialog box help.

- Defaults for all boolean entities is `nil`.
- Due to Valor issue `suppressShapeFill` is always `nil` when using Gerber 4x or 6x.
- If the value of the `drawHolesOnly` parameter is set to `t`, drill holes are drawn for all pads defined on the VIA and PIN CLASS for the film.
- Enable `axlDebug` for additional error messages.

## Arguments

<i>t_filmname</i>	Film name
<i>f_width</i>	Undefined line width, default is 0.
<i>x_sequence</i>	For PDF output ordering. Default is 1, range is 1 to 255. If films have the same number, their database order will determine output.
<i>x_angle</i>	Film rotations, values are 0, 90, 180 or 270, Default is 0.
<i>f_x, f_y</i>	Film offset in design units, Default is 0,0
<i>f_shapeBoundingBo</i>	Shape bounding box in design units. Default is 0.
<i>x</i>	
<i>lt_layers</i>	List of Allegro layers to apply to film. Default is none. Layer names are fully qualified (include both class and subclass)  Example: "ETCH/TOP"  A mode exists where if you specify the class name all subclasses of that class are listed in the film.
	Example: "MANUFACTURING"
<i>lt_domains</i>	List of domains where film should be visible. Values are ipc2581, pdf, artwork and visibility. Default is all.
<i>lt_ipcDomains</i>	List of domains where film should be used in IPC2581. Valid values are inner, outer, misc, doc, and soldermask.

## Value Returned

`t` if film is created, `nil` in case of an error.

## See Also

[axlGetParam](#), [axlDeleteObject](#), [axlDebug](#)

## Examples

- Add/Change

## **Allegro SKILL Reference**

### Parameter Management Functions

---

To understand how to add films, right-click on the artwork dialog to save a film to film (FILM\_SETUP.txt)

- Get all films:

```
p = axlGetParam("artwork")
p->groupMembers
```

- Get a single film (where format is "artwork:<film name>"):

```
s = axlGetParam("artwork:top")
s->??
```

- Delete a film (in this case top):

```
axlDeleteObject(s)
```

## **axlImportXmlDBRecords**

```
axlImportXmlDBRecords(  
    t_fileName  
) -> t/nil
```

### **Description**

This command imports an Allegro Parameter file into the current design. It offers the same capability as (*File – Import – Parameter*). A side effect is creation of `param_read.log` file.



#### **Caution**

***For new releases, the `prm` files may require updating to support new parameter records or additions to current records.***



#### **Tip**

You can create your own custom parameter files and load them with this interface. While the export interface typically groups several Allegro parameters together, you can custom craft a `prm` file with a single parameter record or just a single parameter from one record (see the following example). The only `prm` file requirements are: `prm` file xml header, parameter header and trailer, and revision number per parameter (currently these are all 1)

### **Arguments**

<code>t_fileName</code>	Name of parameter file. Default extension is <code>.prm</code> . If filename is not provided component will located the file via PARAMPATH.
-------------------------	---

### **Value Returned**

`t` if success, `nil` an error

### **See Also**

[axlExportXmlDBRecords](#)

## Examples

See [axlExportXmlDBRecords](#)

Example of a parameter file with setting just the dynamic shape min area to 75.0:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<CadenceAllegroParameter xmlns="">

  <dynfill_parm_type>
    <rev>1</rev>
    <min_area>75.0 MIL</min_area>
  </dynfill_parm_type>
</CadenceAllegroParameter>
```

## **axlMiniStatusReset**

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

### **Description**

This resets the Option panel settings and find filter settings to a new design's default.



***Do not run this unless advise by Cadence.***

### **Arguments**

None

### **Value Returned**

<i>nil</i>	a command is active
<i>t</i>	panel is reset settings

### **Examples**

```
axlMiniStatusReset()
```

## axlPadSuppressGet

```
axlPadSuppressGet(  
    nil  
)  
==> ll_LayerPadSuppress  
  
axlPadSuppressGet(  
    'all  
)  
==> ll_LayerPadSuppress  
  
axlDBGridGet(  
    t_layer/x_layerNumber  
)  
==> l_LayerPadSuppress
```

### Description

Returns pad suppress layer characteristic for a layer or design. Pad suppression is not available in symbol editor.

Name dielectric layers will appear in the list unlike the pad suppress dialog.

### Arguments

<i>nil</i>	Returns global status for dynamic pad suppression
<i>all</i>	Returns all layers with settings
<i>t_layer</i>	Get suppress characteristics of named layer
<i>x_layerNumber</i>	Layer number (1st layer is 0).

### Value Returned

- *ll\_LayerPadSuppress* - list of *l\_LayerPadSuppress* for all etch layers. Layers are ordered from top to bottom.
- *l\_LayerPadSuppress* - suppress characteristics of named layer. The symbols pin and via are optional and if present indicate pin and/or vias will be suppressed on that layer.

(<*t\_layer*> [<*s\_pin*>] [<*s\_via*>])

## See Also

[axlPadSuppressSet](#), [axlPadSuppressOkLayer](#), [axlDBControl](#), [axlSubclassRoute](#),  
[axlPadOnLayer](#)

## Example

- Get the global dynamic pad suppression status

```
suppress = axlPadSuppressGet(nil)
```

- Get and print suppress state of all layers

```
suppress = axlPadSuppressGet('all)  
foreach(item suppress  
    printf("Layer=%s  what= %L\n", car(item) cdr(item))
```

- Get settings for layer "GND"

```
suppress = axlPadSuppressGet("GND")
```

- Get settings for layer 1

```
suppress = axlPadSuppressGet(1)
```

## **axlPadSuppressOkLayer**

```
axlPadSuppressOkLayer(  
    t_layer/x_layerNumber  
)  
==> t/nil
```

### **Description**

Indicates if layer can be set for pad suppression. Only internal conductor and shape layers that are not set for negative artwork, support pad suppression.

### **Argument**

<i>t_layer</i>	name of layer (for example, "TOP")
<i>x_layerNumber</i>	layer number (starts at 0);

### **Value Returned**

*t* if layer can allows pad suppress; *nil* otherwise

### **See Also**

[axlPadSuppressGet](#)

### **Examples**

The following are the same in the PCB tool but may not be in APD+:

```
axlPadSuppressOkLayer("TOP")  
axlPadSuppressOkLayer(0)
```

## axlPadSuppressSet

```
axlPadSuppressSet(
  g_mode
  ll_LayerPadSuppress/'all/'none/nil
)
==> t/nil

axlPadSuppressSet(
  g_mode
  t_layer/x_layerNumber
  ls_options
)
==> t/nil
```

### Description

This modifies the pad suppression settings in the design. Allows control of both dynamic suppression setting (`g_mode`) and the individual layer options (subsequent arguments).

### Notes:

- ❑ If passing a list of suppression layers then any errors in the list are ignored.
- ❑ Will mark dynamic shapes and DRC out of date.
- ❑ If enabling dynamic mode and no suppression layers are enabled the dynamic mode will be left disabled.
- ❑ Unlike the dynamic it will not automatically enable the display of padless holes.



***Pad suppression dialog should not be open when using this API.***

## Argument

*g\_mode*                          The possible values are:

- `nil` - maintain current pad suppression mode
- `'on` - turn pad suppression on
- `'off` - turn pad suppression off

In the first format, second argument can have one of the following values.

`'all`                          Enable suppress on all supported layers

`'none`                        Clear suppression on all supported layers

`nil`                            Leave suppression layers alone (typically used to toggle global mode)

`ll_LayerPadSuppre ss`      List of layers using the same form as [axIPadSuppressGet](#).

Alternatively, use the second format to set suppression on single layers.

`t_layer`                    Layer name

or

`x_layerNumber`            Layer number when first layer is 0

`ls_options`                May be `nil` or a list of `'via` and/or `'pin`

## Value Returned

`t` if success, `nil` a failure

## See Also

[axIDBGridGet](#), [axIDRCUpdate](#), [axIDBDynamicShapes](#)

## Examples

- Enable dynamic suppression setting

## Allegro SKILL Reference

### Parameter Management Functions

---

```
axlPadSuppressSet('on nil)
```

- Enable all layers and dynamic mode

```
axlPadSuppressSet('on 'all)
```

- Delete suppression layer settings and turn off dynamic mode

```
axlPadSuppressSet('off 'none)
```

- Turn on via suppression on layer GND

```
axlPadSuppressSet(nil "GND" '(via))
```

- Turn on via & pin suppression on layer GND

```
axlPadSuppressSet(nil "GND" '(via pin))
```

- Turn off suppression on a layer GND

```
axlPadSuppressSet(nil "GND" nil)
```

- Turn on suppression for GND and VCC layers

```
axlPadSuppressSet(nil '(("GND" via pin) ("VCC" via pin)))
```

## **axlParamFilletDoc**

```
p = axlGetParam("fillet")
axlSetParam(p)
```

### **Description**

This function supports access to the fillet parameter record. The database is updated when axlSetParam is called.

1. This parameter is not available in certain tiers of Allegro PCB Editor.
2. If dynamic is enabled, or if a parameter is changed while dynamic fillet is in effect, when axlSetParam is called, all fillet/tapes are updated.
3. If one of the min/max attributes is changed, the opposite value may be updated to enforce the min <= max rule.

### ***Fillet Attributes***

<b>NAME</b>	<b>Set?</b>	<b>TYPE</b>	<b>DESCRIPTION</b>
objectType	No	string	“fillet”
dynamic	Yes	t/nil	enables/disables dynamic fillet or taper
allowDRC	Yes	t/nil	allows fillet or taper to cause DRC
allowCurved	Yes	t/nil	generates curves for fillet or taper
unusedNets	Yes	t/nil	generates fillet or tapers on unused nets
pin	Yes	t/nil	generates fillet on pins
via	Yes	t/nil	generates fillet on vias
ts	Yes	t/nil	generates fillet on Ts
fingers	Yes	t/nil	generates fillet on fingers
(APD+ only)			
padsWithoutDrills	Yes	t/nil	generates fillet on pads without drills

**Allegro SKILL Reference**  
Parameter Management Functions

---

NAME	Set?	TYPE	DESCRIPTION
round	Yes	t/nil	generates fillet on round pads
square	Yes	t/nil	generates fillet on square pads
rect	Yes	t/nil	generates fillet on rectangular pads
oblong	Yes	t/nil	generates fillet on oblong pads
octagon	Yes	t/nil	generates fillet on octagon pads
padShape	Yes	t/nil	generates fillet on shape pads
sizeRound	Yes	float	minimum size of round pad to fillet
sizeSquare	Yes	float	minimum size of square pad to fillet
sizeRect	Yes	float	minimum size of rectangular pad to fillet
sizeOblong	Yes	float	minimum size of oblong pad to fillet
sizeOctagon	Yes	float	minimum size of octagon pad to fillet
pinDesiredAngle	Yes	integer	desired fillet angle for a pin (0 to 99 degrees)
viaDesiredAngle	Yes	integer	desired fillet angle for a via (0 to 99 degrees)
tDesiredAngle	Yes	integer	desired fillet angle for a T (0 to 99 degrees)
pinMaxAngle	Yes	integer	Maximum fillet angle for a pin (0 to 99 degrees)
viaMaxAngle	Yes	integer	Maximum fillet angle for a via (0 to 99 degrees)
tMaxAngle	Yes	integer	Maximum fillet angle for a T (0 to 99 degrees)
pinMaxOffset	Yes	float	Maximum offset length for a pin
viaMaxOffset	Yes	float	Maximum offset length for a via
tMaxOffset	Yes	float	Maximum offset length for a T

## Allegro SKILL Reference

### Parameter Management Functions

---

<b>NAME</b>	<b>Set?</b>	<b>TYPE</b>	<b>DESCRIPTION</b>
pinMaxArcOffset	Yes	float	Maximum arc offset length for a pin (only if curved fillets are allowed)
pinMinArcOffset	Yes	float	Minimum arc offset length for a pin (only if curved fillets are allowed)
viaMaxArcOffset	Yes	float	Maximum arc offset length for a via (only if curved fillets are allowed)
viaMinArcOffset	Yes	float	Minimum arc offset length for a via (only if curved fillets are allowed)
pinMinLineWidth	Yes	float	Minimum line width from a fillet to a pin
viaMinLineWidth	Yes	float	Minimum line width from a fillet to a via
tMinLineWidth	Yes	float	Minimum line width from a fillet to a T
pinMaxLineWidth	Yes	float	Maximum line width from a fillet to a pin
viaMaxLineWidth	Yes	float	Maximum line width from a fillet to a via
tMaxLineWidth	Yes	float	Maximum line width from a fillet to a T
taper	Yes	t/nil	enable/disable taper generation
taperAngle	Yes	integer	desired taper angle (line to line)
taperMaxOffset	Yes	float	maximum offset length for a line

### Arguments

- axlGetParam*                          Requires “fillet”  
*axlSetParam*                          Requires return of *axlGetParam*

### Value Returned

- *axlgetParam* returns fillet parameter record

- axlSetParam returns parameter dbid if successful, nil otherwise

## Examples

Enable dynamic fillet

```
p = axlGetParam("param")
p->dynamic = t
axlSetParam(p)
```

## See Also

[axlGetParam](#), [axlSetParam](#)

## **axlParamShapeDoc**

```
d= axlGetParam("shapeDynamic")
l= axlGetParam("shapeDynamic:etch/shapeDynamicLayer:<name>")
s= axlGetParam("shapeStatic")
axlSetParam(d)
```

### **Description**

Supports access to the dynamic, static, and layer parameter records. Database is updated when [axlSetParam](#) is called.

#### ***Shape Dynamic Attributes***

NAME	Set?	TYPE	DESCRIPTION
objectType	No	string	"shapeDynamicLayer:<name>"
artwork	Yes	string	Target Gerber format. Values are "Gerber 4x00", "Gerber 6x00", "Gerber RS274X", "Barco DPF", "MDA", or "Non-Gerber"
minAperture	Yes	float	Minimum aperture for shape fill. Value must at least 3 database units (if you have a 2 decimal-place design, this value should be greater than or equal to 0.03).
areaSuppress	Yes	float	Length of a minimum shape to suppress. This value is squared to determine the minimum area to suppress.

## Allegro SKILL Reference

### Parameter Management Functions

---

<b>NAME</b>	<b>Set?</b>	<b>TYPE</b>	<b>DESCRIPTION</b>
inlinePinVoid	Yes	float	Distance between pins to combine into a one void.  If 0, the inline pin void algorithm is disabled.
			In the dialog this value is shown as "Individually".
acuteAngleTrim	Yes	string	Type of trimming. Values are "Round", "Chamfered", or "Full Round".
diffpair	Yes	t/nil	Combined void for vias added with Return Path option.
fillStyle	Yes	string	"Fill style of shape". Values are "Solid" or "XHatch"
xHatchBorderWidth	Yes	float	Width of border.  Must be greater than or equal to the width of the largest of the first or the second hatch line widths.
xHatchLineWidth1	Yes	float	Line width for fill pattern 1
xHatchLineSpacing1	Yes	float	Line spacing for fill pattern 1
xHatchLineAngle1	Yes	float	Line angle for fill pattern 1 (in degrees)
xHatchLineWidth2	Yes	float	Line width for fill pattern 2
xHatchLineSpacing2	Yes	float	Line spacing for fill pattern 2
xHatchOrigin	Yes	list	Origin of the cross-hatch lines. Must contain float values representing x and y coordinates
xHatchLineAngle2	Yes	float	Line angle for fill pattern 2 (in degrees)
xHatchSnapVoids	Yes	t/nil	Whether to attach voids to cross hatch lines.

**Allegro SKILL Reference**  
Parameter Management Functions

---

NAME	Set?	TYPE	DESCRIPTION
xHatchFillPartial	Yes	string	Partial filling of cross-hatch cells. Values are "Off", "Low", "Medium" "High".
pinClearanceOther	Yes	t/nil	Clearance mode; uses DRC for nil and uses thermal/antipad clearance values for, t.
			If nil, the thermalWidthOverSize value is added to line width constraint.
pinOversize	Yes	float	Additional clearance applied to thru pin to shape clearance or thru pin to thermal/antipad clearance.
pinThermalType	Yes	string	Thermal relief type. Can be, "Diagonal", "Orthogonal", "Full contact", "8 way connect" "Orthogonal", "Full contact", "8 way connect" or "None".
pinBestFit	Yes	t/nil	Generative thermals using the best contact option.
pinMinTherms	Yes	integer	Minimum thermal connections. Range is 1-4 unless 8 way connect is used when it is 1-8.
pinMaxTherms	Yes	integer	Maximum thermal connections. Range is 1-4 unless 8 way connect is used when it is 1-8.
smdClearanceOther	Yes	t/nil	Clearance mode; uses DRC for nil and uses thermal/antipad clearance values for t
smdOversize	Yes	float	Additional clearance applied to smd pin to shape clearance or smd pin to thermal/antipad clearance

## Allegro SKILL Reference

### Parameter Management Functions

---

<b>NAME</b>	<b>Set?</b>	<b>TYPE</b>	<b>DESCRIPTION</b>
smdThermalType	Yes	string	Thermal relief type. Can be, "Diagonal", "Orthogonal", "Full contact", "8 way connect" or "None".
smdBestFit	Yes	t/nil	Generative thermals using the best contact option.
smdMinTherms	Yes	integer	Minimum thermal connections. Range is 1-4 unless 8 way connect is used when it is 1-8.
smdMaxTherms	Yes	integer	Maximum thermal connections. Range is 1-4 unless 8 way connect is used when it is 1-8.
viaClearanceOther	Yes	t/nil	Clearance mode; uses DRC for nil and uses thermal/antipad clearance values for t/
viaOversize	Yes	float	Additional clearance applied to via to shape clearance or via to thermal/antipad clearance
viaThermalType	Yes	string	Thermal relief type. Can be, "Diagonal", "Orthogonal", "Full contact", "8 way connect" or "None".
viaBestFit	Yes	t/nil	Generative thermals using the best contact option.
viaMinTherms	Yes	integer	Minimum thermal connections. Range is 1-4 unless 8 way connect is used when it is 1-8.
viaMaxTherms	Yes	integer	Maximum thermal connections. Range is 1-4 unless 8 way connect is used when it is 1-8.
lineOversize	Yes	float	Clearance oversize for lines and clines
textOversize	Yes	float	Clearance oversize for text
shapeOversize	Yes	float	Clearance oversize shapes and fill rectangles.

## Allegro SKILL Reference

### Parameter Management Functions

---

<b>NAME</b>	<b>Set?</b>	<b>TYPE</b>	<b>DESCRIPTION</b>
thermalWidthUseXHatch	Yes	t/nil	Use cross-hatch line widths when generating thermals (applies to only cross-hatch shapes)
thermalWidthUseFixed	Yes	t/nil	Uses thermalWidthFixed value for thermal width for t and adds the thermalWidthOverSize value to line width constraint for nil.
thermalWidthFixed	Yes	float	Fixed width of thermals.
thermalWidthOverSize	Yes	float	Thermal width oversize value added to line width constraint. It is applied when thermalWidthUseFixed is nil.

### ***Shape Static Attributes***

<b>NAME</b>	<b>Set?</b>	<b>TYPE</b>	<b>DESCRIPTION</b>
objectType	No	string	“shapeStatic”
minAperture	Yes	float	Minimum aperture for shape fill. Must be at least 3 database units (if you have a 1 decimal place design this value should be 0.3).
areaSuppress	Yes	float	Length of a minimum shape to suppress. This value is squared to determine the minimum area to suppress.
allowSplit	Yes	string	Allow a shape to split when voiding. Permitted values are "No", "Yes" or "Confirm".
inlinePinVoid	Yes	t/nil	Create single void for in-line pins (t) or individual voids (nil).

**Allegro SKILL Reference**  
Parameter Management Functions

---

NAME	Set?	TYPE	DESCRIPTION
acuteAngleTrim	Yes	string	Type of trimming. Values are "Round", "Chamfered", "Full Round".
xHatchSnapVoids	Yes	t/nil	Whether attach voids to xhatch lines.
xHatchFillPartial	Yes	string	Fill partial xhatch cells. Values are "Off", "Low", "Medium" "High". This setting is shared with dynamic shapes.
pinClearanceType	Yes	string	Clearance mode. Can be "Thrm/Anti", "DRC Value", "Default" or "None".
pinDefault	Yes	float	Clearance value to use if clearance type is "Default".
pinThermalType	Yes	string	Thermal relief type. Can be, "Diagonal", "Orthogonal", "Full contact", "8 way connect".
viaThermalType	Yes	string	Thermal relief type. Can be, "Diagonal", "Orthogonal", "Full contact", "8 way connect".
lineClearanceType	Yes	string	Clearance mode. Can be "DRC Value", "Default" or "None".
lineDefault	Yes	float	Clearance value to use if clearance type is "Default"
clineClearanceType	Yes	string	Clearance mode. Can be "DRC Value", "Default" or "None".
clineDefault	Yes	float	Clearance value to use if clearance type is "Default"
textClearanceType	Yes	string	Clearance mode. Can be "DRC Value", "Default" or "None".
textDefault	Yes	float	Clearance value to use if clearance type is "Default"
shapeClearanceType	Yes	string	Clearance mode. Can be "DRC Value", "Default" or "None".

## Allegro SKILL Reference

### Parameter Management Functions

---

NAME	Set?	TYPE	DESCRIPTION
shapeDefault	Yes	float	Clearance value to use if clearance type is "Default"
rectClearanceType	Yes	string	Clearance mode. Can be "DRC Value", "Default" or "None".
rectDefault	Yes	float	Clearance value to use if clearance type is "Default"
thermalMax	Yes	integer	Maximum thermals to generate by connection, range is 1-8
thermalWidth	Yes	float	Thermal width to use. If 0 uses DRC line width value.

#### Notes

- If not specified all float units are in design units.
- The type of cross hatching is not directly controlled by these two parameter records. Instead, if the cross-hatch values are not supplied at the time of shape creation, the values stored in these parameter records are used.
- If the dynamic shape parameter record is modified any dynamic shapes are marked out of date.
- If the static shape parameter record is modified no existing static shapes are affected. These values are applied when the shape is voided next time.
- Use `axlDBControl ('dynamicFillMode')` to control dynamic shape fill mode.
- Use "cdn\_outer", "cdn\_inner\_plane", and "cdn\_inner\_signal" to access superlayer parameter records.

#### Arguments

`axlGetParam`                    Requires "shapeStatic" or "shapeDynamic"  
`axlSetParam`                    Requires return of `axlGetParam`

#### Value Returned

- `axlGetParam` returns shape parameter record
- `axlSetParam` returns parameter dbid if successful, `nil` otherwise

## See Also

[axlGetParam](#), [axlSetParam](#), [axIDBControl](#), [axIDBDynamicShapes](#)

## Examples

### ■ Set smd thermal for dynamic

```
d = axlGetParam("shapeDynamic")
axlPrintDbid(d)           ; print dbid in a pleasing format
d->smdThermalType = "8 way connect"
d->smdBestFit = t
d->smdMinTherms = 2
d->smdMaxTherms = 7
axlSetParam(d)
```

### ■ Set minimum aperture and allow shapes to automatically split

```
s = axlGetParam("shapeStatic")
s->minAperture = 10.2
s->allowSplit = "Yes"
axlSetParam(s)
```

## axlGetParam

```
axlGetParam (
  t_parm_name
)
⇒ o_paramDbid/nil

axlGetParam (
  nil
) =>lt_params
```

### Description

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

## Arguments

*nil* Returns list of parameters supported

## Allegro SKILL Reference

### Parameter Management Functions

---

*t\_parm\_name* Name of the parameter to seek. The legal naming conventions follow:

- `paramTextBlock:<#>` -- where # is 1-<N> (Example: `paramTextBlock:1`) where N is number of text blocks (`axlDBGetTextBlockCount()`)
- `paramDesign`
- `paramDisplay`
- `paramLayerGroup:<name>` – where name is a legal Allegro class name
- `paramLayerGroup : ETCH` – is obsolete for getting the cross-section layers, use new `axlXSectionGet()` family of APIs. If the design does not contain multiple cross-sections, this will still return the list of ETCH layers, and if there are no mask layers, this will be the list of all layers. It will be maintained for older SKILL code to continue to work in single stackup designs with no mask layers.

**Note:** In IC packaging products a pseudo class called WIRE with a single subclass called WIRE exists. This supports bondwires and is typically not displayed in the Options panel drop-down. Also, predefined ETCH subclasses BOND\_TOP and BOND\_BOTTOM, used for wirebond bondpads on chip-on-board components, are not included as they are not layers available for shapes, lines or routing. See switch `includeNonLayers` that controls this.

- `paramLayerGroup:<name>/paramLayer:<name>`
- `paramLayerGroup : name / includeNonLayers` – ETCH class includes all non-mask layers of the cross-section. By default, `paramLayerGroup : ETCH` includes only those ETCH subclasses that are also cross-section layers. In order to get additional ETCH subclasses that are not layers, such as BOND\_TOP and BOND\_BOTTOM for chip-on-board, add switch `includeNonLayers`.
- `artwork` – List of film names
- `artwork:<filmName>` – A film given by `filmName`
- `testprep` – See `axlParamTestPrepDoc`
- `Fillet` – See `axlParamFilletDoc`
- `shapeStatic` and `shapeDynamic` – See `axlParamShapeDoc`

## Allegro SKILL Reference

### Parameter Management Functions

---

#### Value Returned

o_paramDbid	Returns a <i>dbid</i> for the requested parameter
lt_params	Returns list of parameter names supported
nil	Returns if parameter requested is not legal

#### See Also

[axlSetParam](#), [axlGetParam](#), [axlIsParamType](#), [axlParamDesignDoc](#)

#### Example

- Return all param types supported

```
axlGetParam(nil)
```

- Get etch layer (to find all members of the etch class).

```
etch_parm = axlGetParam("paramLayerGroup:ETCH")
```

```
param:123456
```

```
etch_parm->??  
(objType "paramLayerGroup" name "ETCH" visible  
-1 nChildren 4 groupMembers  
("TOP" "GND" "VCC" "BOTTOM")  
color -1  
)
```

```
etch_parm->color  
-1
```

```
etch_parm->groupMembers  
("TOP" "GND" "VCC" "BOTTOM")
```

- Access artwork records

- Get list of all possible records.

```
p = axlGetParam("artwork")  
p->??  
(objType "artwork" nChildren 4 groupMembers  
("TOP" "GND" "VCC" "BOTTOM")
```

## Allegro SKILL Reference

### Parameter Management Functions

---

)

- Get information on film record “VCC”.

```
r = axlGetParam("artwork:VCC")
r->??
(objType "artwork" groupMembers
("ETCH/VCC" "PIN/VCC" "VIA CLASS/VCC") vectorBasedPad
t suppressShapeFill t useApertureRotation nil
drawMissingPadApertures nil suppressUnconnectPads t fullContact
nil mirrored nil shapeBoundingBox 100.0
offset (0.0 0.0) rotation 0 undefineLineWidth
0.0 negative t name "VCC"
)
```

- Delete a TOP parameter record.

```
axlDeleteObject(axlGetParam("artwork:TOP"))
```

- Design (paramDesign) modification.

```
axlDBChangeDesignOrigin: change design origin
axlDBChangeDesignExtents: change extents
axlDBChangeDesignUnits: change units and/or accuracy
```

## **axlParamDesignDoc**

```
axlGetParam("paramDesign") -> o_dbid
```

### **Description**

This function obtains the Allegro design record which controls database size and accuracy.

### **DESIGN Attributes**

NAME	Set?	TYPE	DESCRIPTION
accuracy	no	integer	number of decimal places in design
dbuperuu	no	integer	database units per user units in design
bBox	no	bbox	the design's bounding box
height	no	float	size of design (see width)
objType	no	string	"paramDesign"
units	no	string	design units ("mils", etc.)
width	no	float	size of design (see height)
xy	no	point	lower left corner of design

### **Arguments**

*"paramDesign"*      Name of parameter

### **Value Returned**

dbid of parameter record

### **Example**

See what current db has to offer

```
p = axlGetParam("paramDesign")
p->??
```

## **Allegro SKILL Reference**

### Parameter Management Functions

---

#### **See Also**

[axIDBChangeDesignOrigin](#), [axIDBChangeDesignExtents](#), [axIDBChangeDesignUnits](#)

## axlSetParam

```
axlSetParam (
  od_paramDbid
)
⇒ rd_paramDbid/nil
```

### Description

This allows applications to modify certain aspects of Allegro parameters. After a parameter has been retrieved, attributes of it can be changed locally. Those changes can then be put back into the database using `axlSetParam`.

### Arguments

*od\_paramDbid*      Parameter id returned from [axlGetParam](#). Modify the parameters to be changed then call `axlSetParam` function to update the database.

### Value Returned

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

### Example

#### ■ Change visibility (note it is easier to use [axlVisibleSet](#) to do this)

```
(setq etch_top (axlGetParam "paramLayerGroup:ETCH/paramLayer:TOP"))
=>param:123456
; is layer visible?
etch->visible
t
; blank it
etch_top->visible = nil
t
(axlSetParam etch_top)
=>param:123456
; layer is now invisible
etch_top->visible
```

## **Allegro SKILL Reference**

### Parameter Management Functions

---

nil

#### ■ **Change accuracy**

```
p = axlGetParam("paramDesign")
p->accuracy = 3
axlSetParam(p)
```

## **axlIsParamType**

```
axlIsParamType(  
    o_paramDbid  
) -> t/nil
```

### **Description**

Returns *t* if the parameter is a parameter dbid, else return *nil*.

### **Arguments**

A parameter type from [axlGetParam](#).

### **Value Returned**

<i>t</i>	if a parameter
<i>nil</i>	otherwise

### **See Also**

[axlGetParam](#)

### **Example**

```
axlIsParamType( axlGetParam("shapeDynamic") )  
-* /
```

## **axltdfIsEnabled**

```
axltdfIsEnabled()  
) -> t/nil
```

### **Description**

This function checks whether a technology-dependent footprint mapping file (.tdfx) is attached to a design. This function also checks whether the technology-dependent footprint mapping, defined in the TDF file, contains mapping of the footprint symbols to alternative footprints.

### **Arguments**

None.

### **Value Returned**

t	if TDF information is attached
nil	no attachment

### **See Also**

[axltdfSave](#), [axltdfLoad](#)

### **Examples**

```
if(axltdfIsEnabled()  
println("TDF mapping found")  
println("No TDF mapping in this drawing")  
)  
=> "TDF mapping found"
```

## **axltdfLoad**

```
axltdfLoad  
  [t_tdffile]  
) -> t/nil
```

### **Description**

Parses the input TDF file to get the technology-dependent footprint mapping and stores it in the database.

### **Arguments**

t\_tdffile              Optional name of TDF file.

If no name is given then the currently attached TDF mapping is deleted and symbols are refreshed to reflect the changes.

### **Value Returned**

t              If successful

nil              If failed

### **See Also**

[axltdfSave](#), [axltdfIsEnabled](#)

### **Examples**

- Imports the TDF file `tdf_example.tdfx`

```
axltdfLoad("tdf_example")  
==> t
```

- Deletes the currently attached TDF file

```
axltdfLoad()  
==> t
```

## **axltdfSave**

```
axltdfSave  
  [t_tdffile]  
  ) -> t/nil
```

### **Description**

Saves current technology-dependent footprint mapping to the file name supplied as the argument (or to default of `tdf_attachment.tdfx` if none is provided).

### **Arguments**

`t_tdffile`                   Optional name of TDF file.

### **Value Returned**

<code>t</code>	If successful
<code>nil</code>	If failed

### **See Also**

[axltdfLoad](#), [axltdfIsEnabled](#)

### **Examples**

- Saves TDF attachment to `exported.tdfx`  

```
axltdfSave ("exported")  
==> t
```
- Saves TDF attachment to default name (`tdf_attachment.tdfx`)  

```
axltdfSave ()  
==>t
```

## Color Access

### **axlColorDoc**

`axlColorDoc`

#### **Description**

Allegro supports two color access methods: pre-defined colors and Allegro database colors. Not all Allegro-based programs support access to Allegro database colors. (This is only supported by the graphics editors.)

Pre-defined colors are set and accessed by their symbols:

- `'black`
- `'white`
- `'red`
- `'green`
- `'yellow`
- `'blue`
- `'cyan`
- `'magenta`
- `'darkred`
- `'darkgreen`
- `'darkblue`
- `'darkcyan`
- `'darkmagenta`
- `'darkyellow`
- `'multivalue - use dfor fields where value is not the same`
- `'button - current color of button faces (grey)`
- `'default - default background or text color per application theme`
- `'modified - color used to denote modified fields`

## Allegro SKILL Reference

### Parameter Management Functions

---

In addition, graphics editors support access to the colors used for Allegro layers. These are integer numbers.

AXL API calls such as `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.

Example:

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

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

Interfaces supporting setting color are mostly form based. For there interfaces see:

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

#### Notes

- No AXL method is currently supported to allow you to change the red/green/blue (RGB) of Allegro database colors
- We restrict the pre-defined colors to those defined to minimize use of colors to minimize problems with 8 bit color graphics on UNIX. When 24 (or higher) color cards become standard on UNIX, this will be relaxed.
- On Windows, Microsoft's UI theme overrides the background color. To enable background color control for ENUM controls, when specifying the control in the form file add the "OPTION color". The default Microsoft theme for this control is disabled. Also, the drop-down itself drawn with the background color.

On UNIX, this option is ignored and background coloring just works.

#### Arguments

none

## **Allegro SKILL Reference**

### Parameter Management Functions

---

#### **Value Returned**

none

## axlColorGet

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

### 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.
- Returns number of patterns supported (includes default solid).

The color index is the number assigned to each layer in Allegro PCB Editor. (see axlVisibleGet).

## Arguments

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

## Value Returned

<i>x_count</i>	Size of database palette.
nil	Error.
<i>lx_rgb</i>	A palette.
<i>lx_all</i>	The entire database palette.

## See Also

[axlColorSet](#), [axlVisibleGet](#)

## 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)
```

Get number of display patterns supported

```
cnt = axlColorGet(`pattern)
```

## axIColorShadowGet

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

### Description

Provides the options of shadow mode.

### Arguments

<i>g_option</i>	
'mode	Shadow mode status ( <i>t</i> is on, <i>nil</i> is off).
'activeLayer	Active layer dimming enabled ( <i>t</i> ). This is called "Dim active layer in Options panel.
'highlight	This is called "Dim color assignments" in the Options panel
'percent	Current brightness percentage (0 to 100).
'custom	Custom colors, these are not shadowed.

### Value Returned

<i>t/nil</i>	Shadow or active layer mode on or off.
<i>x_percent</i>	Brightness percentage.

### See Also

[axIColorSet](#), [axIColorShadowSet](#)

### Examples

Is shadow mode on:

```
axIColorShadowGet('mode)
```

Is shadow mode percent:

## **Allegro SKILL Reference**

### Parameter Management Functions

---

`axlColorShadowGet('percent')`

## axlColorShadowSet

```
axlColorShadowSet(  
    g_mode  
    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.

The Mode 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. This is called "Dim active layer in Options panel."
- The highlight can be `t` or `nil` to dim highlighted objects. This is called *Dim color assignments* in the Options panel.
- 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. For example, you have what appears to be black on black.

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

*g\_mode*

The possible values are:

- 'mode – Enable or disable shadow mode.
- 'highlighted – Enable or disable shadow mode for highlighted objects.
- 'activeLayer – Enable or disable active layer dimming.
- 'percent – Set shadow mode percentage (0 to 100)

## Value Returned

*t* If successful.

*nil* An argument error.

## See Also

[axlColorSet](#), [axlColorShadowSet](#), [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 an Allegro 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.
```

## Allegro SKILL Reference

### Parameter Management Functions

---

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

In Allegro 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 Allegro 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, Allegro 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 – Obsolete**

This function is obsolete. Due to change in display model, switching off colors is no longer supported.

## **axIColorOnSet – Obsolete**

This is an obsolete command. Due to changes in the viewing model, now you cannot turn off a color in Allegro PCB Editor.

## **axIColorPriorityGet – Obsolete**

Due to the changes in color model of Allegro PCB Editor, this command is now obsolete.  
Instead of this command, use [axILayerPriorityGet](#).

## **axIColorPrioritySet – Obsolete**

Due to the changes in color model of Allegro PCB Editor, this command is now obsolete.  
Instead of this command, use [axILayerPrioritySet](#).

## **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**

<i>t</i>	Successful.
nil	Failed to save.

## **EXAMPLES**

Save current design color settings:

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

### **See Also**

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

## axlColorSet

```
axlColorSet(  
    x_number / 'background  
    l_rgb  
) -> 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 intensity) to 255 (maximum intensity).

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

### ***Color model:***

A color (or colorNumber) in Allegro 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.
- Each color number can be assigned to a layer. Multiple layers will have the same color number, because there are more layers than colors.
- Allegro 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.

## **Allegro SKILL Reference**

### Parameter Management Functions

---

axlColorGet	Get red, green, or blue of one or more color numbers.
axlColorShadowGet	Shadow mode options.
axlColorShadowSet	Set shadow mode options.
axlLayerPrioritySet	set a layer to a display priority
axlLayerPriorityGet	get a layer's current priority
axlLayerPriorityClearAll	clear all layer priorities (restore to default)
axlLayerPrioritySaveAll	save existing priority table
axlLayerPriorityRestoreAll	restore saved priority table
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.
axlVisibleGet	Get visibility set for design.
axlVisibleSet	Set visibility set for design.
axlVisibleDesign	Global design visibility control.
axlVisibleUpdate	Update windows with color changes.

## Arguments

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

## Value Returned

<i>t</i>	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)))
```

## **axlCVFColorChooserDlg**

```
axlCVFColorChooserDlg(  
    [x_color_index]  
    [g_show_hilite]  
    [x_hilite_flag]  
    [x_bitmap_index]  
)  
==> t/nil
```

### **Description**

Displays color palette modal dialog. Color wells reflect current design colors.

### **Arguments**

<i>x_color_index</i>	Color index to initialize palette dialog. Values 0 to 191.
<i>g_show_hilite</i>	Specifies whether or not the highlight check box is to be displayed. If the value is set to:  t – displays the highlight check box.  nil/default – highlight check box is not displayed.
<i>x_hilite_flag</i>	Highlight state to initialize highlight check box (if displayed). Pass 1 or 0.
<i>x_bitmap_index</i>	Bitmap index to initialize palette dialog. Values 0 to 15.

### **Value Returned**

<i>list</i>	containing one or two int values for user color palette selection and highlight check box selection. if <i>g_show_hilite</i> is not nil, <i>list</i> contains the two values, or else <i>list</i> contains color index only.
<i>nil</i>	if user cancels the form or error occurred.

## **axlClearObjectCustomColor**

```
axlClearObjectCustomColor(  
    [lo_dbid]  
)  
==> t/nil
```

### **Description**

Clears any custom color set on the provided database ID or list of database IDs. Objects supported are groups, nets, symbol instances, function instances, pins, vias, bond fingers, and external DRCs. The color of an object (group, net, symbol, or function) and all inheriting objects are cleared. If you clear the color of an object, where part of that object, such as a pin in a net, has a different custom color, that child object will retain its different color. If you want the custom color to be cleared for the parent and all child objects of the parent, call `axlClearObjectCustomColor` on the parent object and on all child objects that have different custom colors.

**Note:** Segments may not be custom colored or cleared by these interfaces. Segments are colored directly and only for temporary use in specific commands like the highlight segment over via tool. They should be unhighlighted with the same command.

Custom colors must be enabled (see `axlDBDisplayControl`) to be viewed.

### **Arguments**

<i>lo_dbid</i>	List of database IDs or a single database ID to clear custom color.
----------------	---

### **Value Returned**

<i>t/nil</i>	Returns <i>t</i> if at least one object custom color was cleared. Returns <i>nil</i> otherwise.
--------------	--

### **Examples**

See `axlCustomColorObject` for examples.

## **Allegro SKILL Reference**

### Parameter Management Functions

---

#### **See Also**

[axlCustomColorObject](#)

## **axlCustomColorObject**

```
axlCustomColorObject(  
    [lo_dbid]  
    [g_custom_color]  
)  
==> t/nil
```

### **Description**

Set custom color for a single or list of database IDs. Objects supported are groups, nets, symbol instances, function instances, pins, vias, bond fingers, clines, lines, text, figures, shapes, and external DRCs. An object that is part of a group, net, symbol, or function but that does not have a specific custom color assigned, inherits the custom color assigned to the parent object. For instance, a pin on net "A" inherits the custom color assigned to net "A" if the pin does not have a different custom color assigned to it. On the other hand, if a custom color is assigned to a net but some pins on the net have a different custom color assigned, the color of these pins will not match the net color.

**Note:** Segments may not be custom colored or cleared by these interfaces. Segments are colored directly and only for temporary use in specific commands like the highlight segment over via tool. They should be unhighlighted with the same command.

The color index is between 1 and `axlColorGet(`count)`. The index references a RGB value in the Allegro Color table. The RGB values can be viewed or modified via `axlColorGet` or `axlColorSet`.

Custom colors need to be enabled (see [axlDBDisplayControl](#)) to be viewed.

## Allegro SKILL Reference

### Parameter Management Functions

---

#### Arguments

<i>od_dbid</i>	a single or a list of DBIDS
<i>g_custom_color</i>	Color index to be used to set custom color. If the value is <i>nil</i> , perm highlight will be used.

#### Value Returned

<i>t</i>	An object was custom colored.
<i>nil</i>	No valid DBIDS.

#### See Also

[axlClearObjectCustomColor](#), [axlDBDisplayControl](#), [axlIsCustomColored](#), [axlColorGet](#)

#### Example

The example covered in this section uses `axlCustomColorObject` and `axlClearObjectCustomColor` functions respectively to set and clear custom color of database elements during interactive commands.

The following example does the following:

- Defines the function `highlightLoop`.
- Loops on the function `axlSelect` gathering user selections to set/clear custom color.
- Custom colors objects using color 4.
- Waits then clears custom color.

The command can be stopped at any time by selecting Cancel or Done from the pop-up menu.

```
(defun customColorLoop ()
  axlSetFindFilter( ?enabled '("noall" "alltypes" "nameform")
  ?onButtons "alltypes")
  while( axlSelect()
    axlCustomColorObject( axlGetSelSet() 4)
    checkColor = axlIsCustomColored( car(axlGetSelSet()) )
    axlSleep(1)
    axlClearObjectCustomColor( axlGetSelSet())
  )
)
```

## **axILayerPriorityClearAll**

```
axILayerPriorityClearAll(  
    ) -> t/nil
```

### **Description**

Clears all layer priority information in Allegro database. Use [axILayerPrioritySet](#) for usage.

### **Arguments**

None

### **Value Returned**

*t* : success

### **See Also**

[axILayerPrioritySaveAll](#), [axILayerPriorityRestoreAll](#)

## **axlLayerPriorityGet**

get layer's priority

```
axlLayerPriorityGet(  
    t_layer  
) -> x_priority/t_mapClass/nil
```

### **Description**

Obtains layer priority, where 0 is normal (not set). Priority can range from 1 (highest) to 255 (lowest).

Depending on the argument value, the function operates in two modes:

- if *t\_layer* is layer name (class / subclass), returns priority of that layer as an integer
- if *t\_layer* is class name then returns the mapped layer

**Note:** Mapped layer groupings may change from release to release (for example, future releases may choose to break up some class groupings).

### **Argument**

*t\_layer*      layer name (<class>/<subclass>) or class name (<class>)

### **Value Returned**

- *x\_priority* - priority of layer (0 layer draws at normal priority)
- *t\_mapClass* - class name used as lead group for provided class
- *nil* - error in layer name

### **See Also**

[axlLayerPrioritySet](#)

### **Examples**

- Get and fetch priority

```
axlLayerPrioritySet("BOARD GEOMETRY/OUTLINE" 1)  
prior = axlLayerPriorityGet("BOARD GEOMETRY/OUTLINE")
```

## **Allegro SKILL Reference**

### Parameter Management Functions

---

- Get group class mapping of class Ref Des

```
axlLayerPrioritySet("REF DES") -> "COMPONENT VALUE"
```

## **axlLayerPriorityRestoreAll**

```
axlLayerPriorityRestoreAll(  
    ) -> t/nil
```

### **Description**

Restores previously saved layer priority information. This function only works if a call to [axlLayerPrioritySaveAll](#) has been done already.

### **Arguments**

None

### **Value Returned**

<i>t</i> :	success
<i>nil</i> :	nothing to restore.

### **See Also**

[axlLayerPrioritySaveAll](#), [axlLayerPriorityClearAll](#)

## **axILayerPrioritySaveAll**

```
axILayerPrioritySaveAll(  
    ) -> t/nil
```

### **Description**

Saves all layer priority information to be restored later. Until a [axILayerPriorityRestoreAll](#) is called, any subsequent calls to this function are no-op.

### **Arguments**

None

### **Value Returned**

<i>t</i> :	success
<i>nil</i> :	this function has been called already but axILayerPriorityRestoreAll has not been called yet.

### **See Also**

[axILayerPriorityClearAll](#), [axILayerPriorityRestoreAll](#)

## **axlLayerPrioritySet**

```
axlLayerPrioritySet(  
    t_layer  
    x_priority  
) -> t/nil
```

### **Description**

This changes the drawing priority of given layer. Priority is from 1 (highest) to 255 (lowest). Layers without priority in standard drawing order below all priority layers. The active layer is always drawn first.

Only one layer may be at a priority level, thus adding a new layer at a priority replaces the existing layer at that priority. For example, executing following line of code results in just the ASSEMBLY\_TOP being drawn at priority 1 and OUTLINE returning to normal drawing order.

```
axlLayerPrioritySet( "BOARD GEOMETRY/OUTLINE" 1)  
axlLayerPrioritySet( "PACKAGE GEOMETRY/ASSEMBLY_TOP" 1)
```

From priority level 1 each level must be set for lower priority levels to be enabled. For example, if you set a layer to priority level 2 but leave level 1 empty then level 2 is disabled until level 1 is assigned.

Classes may be grouped together in a class group with one class being the lead of that group. For example, all etch layers (ETCH, PIN, etc.) are mapped together into the stack-up group with class ETCH the lead. You can set the priority using class names but you cannot prioritize the different stack-up layers individually. This interface automatically maps a class name to its class group (see [axlLayerPriorityGet](#) to determine groupings).

You should do a [axlVisibleUpdate](#) after changing layer priority to have the display updated.

**Note:** Priority value of 0 means remove layer priority of the layer.

## Arguments

<i>x_layer</i>	layer name (i.e. "ETCH/TOP")
<i>x_priority</i>	priority value in the range of 1-255 and 0 means remove.

## Value Returned

<i>t</i>	success
<i>nil</i>	error in one of the arguments

## Examples

Set priority for class BOARD GEOMETRY and subclass OUTLINE:

```
axlLayerPrioritySet("BOARD GEOMETRY/OUTLINE" 1)
```

To temporarily force a set of layers to display on top, you should take the following steps:

- save existing layer table,
- clear existing layer priorities
- set your layer priorities
- draw objects
- restore old layer priority:

```
axlLayerPrioritySaveAll()  
axlLayerPriorityClearAll()  
axlLayerPrioritySet() -- multiple times if needed  
axlLayerPriorityRestoreAll()
```

## See Also

[axlLayerPriorityClearAll](#), [axlLayerPrioritySaveAll](#), [axlLayerPriorityRestoreAll](#),  
[axlLayerPriorityGet](#), [axlMapClassName](#), [axlVisibleUpdate](#)

## **axlIsCustomColored**

```
axlIsCustomColored (
  o_dbid
)
==> x_customColor/nil
```

### **Description**

If object has custom color, will return the object custom color, otherwise nil.

### **Arguments**

*o\_dbid* An dbid for which custom color information is desired.

### **Value Returned**

*x\_customColor* custom color or nil if object has no custom color or object does not support custom color.

### **See Also**

[axlCustomColorObject](#)

## Database Layer Management

These functions allow easier access to layer attributes.

### **axlClasses**

```
axlClasses(  
    ) -> lt_classes
```

#### **Description**

Return list of classes. This is actually just:

```
axlGetParam("paramLayerGroup")->groupMembers
```

#### **Arguments**

Nothing

#### **Value Returned**

list of class strings

#### **See Also**

[axlSubclasses](#), [axlGetParam](#), [axlMapClassName](#)

#### **Examples**

```
axlClasses()
```

## **axlCompareLayers**

```
axlCompareLayers(
    l_referenceLayers
    l_compareLayers
    t_outputLayer
    ?action
    ?clearOutputLayer
    ?extents
    ?shape
) => t/nil
```

### **Description**

Performs a comparison between two layers or sets of layers in the active design. This SKILL function provides a direct API to generate layer compare outputs from the code instead of through the interactive and batch layer compare tools.

## Arguments

<i>l_referenceLayers</i>	Source layer(s) to be compared. This may be either a single layer, such as "ETCH/TOP", or a list of layers ('"ETCH/TOP" "PIN/TOP"). If multiple layers are provided, layers are first combined together (logical OR) before the comparison to the second layer set is performed.
<i>l_compareLayers</i>	Comparison layer(s) to be compared. This may be either a single layer, such as "ETCH/TOP", or a list of layers ('"ETCH/TOP" "PIN/TOP"). If multiple layers are provided, these are first combined together (logical OR) before the comparison to the reference layer set is performed.
<i>t_outputLayer</i>	Target layer, where the results of the boolean operation should be generated to. If the layer does not exist, it will be created.
<i>ts_action</i>	Comparison action. Operations are same as those supported by the axlPolyOperation API: 'AND, 'OR, 'ANDNOT, and 'XOR. Default is 'OR.
<i>b_clearOutputLayer</i>	If <i>t</i> , remove any shapes from the output layer before populating with the results of the layer compare operation. Default is <i>t</i> .
<i>l_extents</i>	If set, represents the window area where the comparison is to be restricted to. Default is to compare the entire database region.
<i>g_shape</i>	If set, provides a DBID of a shape. The shape will be used to set the restricted comparison region. Only one of <i>l_extents</i> or <i>g_shape</i> may be used at a time.

## Value Returned

<i>t</i>	Comparison successful
<i>nil</i>	An error occurred. Check arguments passed to the function and ensure that if the output layer does not exist, there is room to add it to the database.

## Example

```
axlCompareLayers("ETCH/TOP" "ETCH/BOTTOM" "DRAWING FORMAT/COMPARE_LAYERS"  
?action 'AND ?clearOutputLayer t)
```

## **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, DIELECTRIC, PLANE, SURFACE, DIESTACK or MASK.

**Note:** See cross-section dialog for a current list.

### **Arguments**

*t\_layername*      Layer name is <class>/<subclass>.

### **Value Returned**

*t\_layertype*      Layer type string.

nil      If an error occurs

### **See Also**

[axlXSectionGet](#)

### **Example**

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

## **axlGetExternalLayers**

```
axlGetExternalLayers(  
    ) -> lt_layers
```

### **Description**

Returns the names external layers.

### **Arguments**

Nothing

### **Value Returned**

(<top name> <bottom name>)

### **See Also**

[axlXSectionGet](#), [axlLayerExternal](#)

### **Examples**

On a PCB design

```
axlGetExternalLayers()  
("TOP" "BOTTOM")
```

## axlGetXSection – Obsolete

```
axlGetXSection(  
)  
==> ll_layers/nil
```

### Description



This is obsolete, use new axIXSectionGet() family of APIs. The command will be maintained for older SKILL code but it will not be enhanced to support new data.  
Returns a list of all layers in the cross section found in the current drawing.

### Values Returned

An ordered SKILL list of layers in the board's cross section. A list of the following format defines each layer:

```
(t_name t_type t_material t_thickness t_thermalCond t_elecCond  
t_dielectricConst y_artworkNeg y_shield t_lossTangent  
t_usage t_SignalDieConstant t_SignalLossTangent g_freqDepFileName)
```

where:

<i>t_name</i>	Layer name.
<i>t_type</i>	Layer type.
<i>t_material</i>	Layer material.
<i>t_thickness</i>	Layer thickness.
<i>t_thermalCond</i>	Layer thermal conductivity.
<i>t_elecCond</i>	Layer electrical conductivity.
<i>t_dielectricConst</i>	Layer dielectric constant.
<i>y_artworkNeg</i>	Indicates whether the artwork for the layer is negative.
<i>y_shield</i>	Indicates whether the layer is a shield layer.
<i>t_lossTangent</i>	Layer loss tangent (valid for dielectrics only).
<i>t_usage</i>	obsolete ("")
<i>t_SignalDieConstant</i>	Dielectric between traces on interior signal layers (or <i>nil</i> ).

## **Allegro SKILL Reference**

### Parameter Management Functions

---

<i>t_SignalLossTangent</i>	Dielectric between traces on interior signal layers (or <i>nil</i> ).
<i>g_freqDepFileName</i>	Defines the name of the frequency-dependent data file for the file; <i>nil</i> if no file name is defined for this layer.
<i>t_etchFactor</i>	Defines the etch factor for this layer which is in degrees.

**Note:** The *t\_SignalDieConstant* and *t\_SignalLossTangent* are *nil* on PLANE and dielectric layers.

## **axlIsEtchLayer**

```
axlIsEtchLayer(  
    t_layer  
)  
=> t/nil
```

### **Description**

Determines if a layer is associated with the ETCH layers. Returns *t* if layer is associated with any of the ETCH layers — ETCH, PIN,VIA, DRC, VIA\_KEEPOUT, ROUTE\_KEEPOUT, ANTIETCH, BOUNDARY, CONSTRAINT\_REGION, ROUTER\_PLAN, and CAVITY

### **Arguments**

<i>t_layer</i>	Layer name (for example, "ETCH/TOP") or just class name ("ETCH")
----------------	--

### **Value Returns**

<i>nil</i>	Not an etch associated layer
<i>t</i>	Is an etch associated layer

### **Examples**

```
axlIsEtchLayer("PIN/TOP")  
axlIsEtchLayer("ETCH")
```

## **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
```

## **axILayerCreateCrossSection – Obsolete**

This function is obsolete, use [axIXSectionCreate](#).

## **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.

## **axlLayerDelete**

```
axlLayerDelete(  
    t_layerName/x_layerNumber  
) => t/nil
```

### **Description**

This command deletes a cross section layer. While [axlDeleteObject](#) can be used to delete empty named layers, this API can delete both named and unnamed cross-section layers. The cross section has both ETCH layers and unnamed dielectric layers. The order of the cross section is returned by [axlGetXSection – Obsolete](#). The `x_layerNumber` is the order within the cross-section with the first index number (e.g AIR) starting at 0.

The command can fail in the following scenarios.

- Deleting a named layer containing geometries (excluding pins or vias)
- Deleting top or bottom dielectric or TOP or BOTTOM etch layers
- Layer name does not exist
- Layer number is less than 0 or greater or equal to  
`length(axlGetXSection('count'))`

### **Arguments**

ETCH layer string or cross section index

### **Value Returned**

`t` if layer is deleted, `nil` if failure

### **Examples**

- The command to delete a layer named empty is:  

```
axlLayerDelete("EMTPY")
```
- Delete the third cross section layer. On most designs this is an unnamed dielectric layer between TOP and the next etch layer.  

```
axlLayerDelete(3)
```

## **Allegro SKILL Reference**

### Parameter Management Functions

---

#### **See Also**

[axILayerCreateNonConductor](#), [axIDeleteByLayer](#), [axIDeleteObject](#), [axIGetXSection – Obsolete](#), [axIXSectionDelete](#)

## **axlLayerExternal**

```
axlLayerExternal(  
    t_subclass/x_subclassNumber  
) -> t/nil
```

### **Description**

Returns if layer is external.

### **Arguments**

<i>t_subclass</i>	Subclass name
<i>x_subclassNumber</i>	The subclass number (Does not mean the cross-section number)

### **Value Returned**

<i>t</i>	External
<i>nil</i>	Not external

### **See Also**

[axlXSectionGet](#), [axlGetExternalLayers](#)

### **Examples**

#### ■ On a design

```
axlGetExternalLayers("TOP") -> t
```

#### ■ On a design

```
axlGetExternalLayers(0) -> t
```

## axlLayerGet

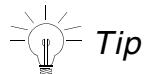
```
axlLayerGet(  
    t_layer  
)  
⇒ o_dbid/nil
```

### Description

Gets the layer parameter given the shortcut notation of <class>/<subclass>. This is an ease of use function that does:

```
axlGetParam("paramLayerGroup:<class>/paramLayer:<subclass>")
```

This function does NOT allow access to the cross-section data (for example, material or thickness). It allows easier access to color and visibility of a layer.



You can use the groupMembers attribute of result —  
`result=axlGetParam( "paramLayerGroup:<class>" )` — to iterate over all subclass of a class.

### Arguments

*t\_layer*    Name of layer in format "<class>/<subclass>".

### Value Returned

<i>o_dbid</i>	Layer parameter <i>dbid</i> .
<i>nil</i>	Layer is not present.

### Example

Changes color of top etch layer.

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

## **Allegro SKILL Reference**

### Parameter Management Functions

---

#### **See Also**

[axlGetParam](#)

## **axILayerRenameNonConductor**

```
axILayerRenameNonConductor (
    t_oldLayerName
    t_newLayerName
)
⇒ t/nil
```

### **Description**

Renames the old subclass to the new name. If the subclass is not found, or if the new name already exists in the design, then no changes will be made.

### **Arguments**

<i>t_oldLayerName</i>	<class>/<subclass>
<i>t_newLayerName</i>	<subclass>

### **Value Returned**

<i>t</i>	If subclass was renamed
<i>nil</i>	Otherwise

### **Example**

```
axILayerCreateNonConductor("BOARD GEOMETRY/FOO")
axILayerRenameNonConductor("BOARD GEOMETRY/FOO" "BAR")
```

### **See Also**

[axILayerCreateNonConductor](#), [axILayerGet](#)

## **axlLayerViaLabel**

```
axlLayerViaLabel(  
    t_layerName/x_layerNumber  
) => t_viaLabel/nil
```

### **Description**

Reports via label for a layer. A via label either defaults to the layer number, or can be assigned by the user through the cross-section. You can query the cross-section for an override.

### **Arguments**

ETCH layer string or cross-section index

### **Value Returned**

Via label name; or `nil`, in case of an error

### **Examples**

- To return the via label of a layer called TOP

```
axlLayerViaLabel("TOP") -> 1
```

### **See Also**

[axlXSectionGet](#)

## axlMaterialGet

```
axlMaterialGet(  
    nil  
)  
==> lt_materials  
  
axlMaterialGet(  
    t_materialName  
)  
==> og_material  
  
axlMaterialGet(  
    'all  
)  
==> log_materials  
  
axlMaterialGet(  
    'file  
)  
==> t_file  
  
axlMaterialGet(  
    'path  
)  
==> t_pathToFile
```

### Description

Returns various information about materials file. Depending on the argument passed, command works in different modes to retrieve the following.

- List of materials
- Number of material file entries
- Name of material file being used; this differ between PCB and IC Packaging products
- Path to material file
- actual attributes of a material file entry

Material attributes as a disembodied property list:

NAME	TYPE	DESCRIPTION
dielectric	double	Dielectric Constant
freqDepFile	string	Name of frequency dependent file or ""

## Allegro SKILL Reference

### Parameter Management Functions

---

NAME	TYPE	DESCRIPTION
electricalConductivity	string	Electrical Conductivity in mho/cm
lossTangent	double	Loss Tangent
name	string	Name of material
objType	string	"material"
readOnly	t	Cannot modify object
thermalConductivity	string	Thermal Conductivity in w/cm-degC
thickness	string	Layer thickness with design units
tolPlus	string	Layer thickness tolerance plus with design units
tolMinus	string	Layer thickness tolerance minus with design units



***On Windows, performance may be slow, when accessing individual material entries in material file is stored on the network.***

## Arguments

<i>nil</i>	List of material names
<i>t_materialName</i>	Name of material
<i>'all</i>	Get all material data
<i>'filename</i>	Get name of material file (Allegro uses 'material' and IC Packaging products uses 'mcmmat'
<i>'path</i>	Return location of file on disk

## Value Returned

<i>nil</i>	An error was detected
<i>lt_materialNames</i>	A list of the material names
<i>og_materials</i>	Disembodied property list of material characteristics (see above)
<i>log_materials</i>	A list of disembodied property lists
<i>t_file</i>	Name of material file in use
<i>t_pathToFile</i>	Path to material file in use

## Examples

### ■ Get info about FR-4

```
mat = axlMaterialGet("FR-4")
printf("Thickness %L\n" mat->thickness)
```

### ■ Get all materials defined in the materials file

```
names = axlMaterialGet(nil)
```

### ■ Get path of file

```
path = axlMaterialGet('path)
```

## **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 visisbility 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.

## See Also

[axlVisibleUpdate](#) and [axlIsVisibleLayer](#)

**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 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. If you want to update the display, call `axlVisibleUpdate` when finished with your layer visibility updates.

### **Arguments**

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

### **Value Returned**

<code>t</code>	Layer set to visible or invisible as specified.
<code>nil</code>	Layer does not exist.

### **See Also**

#### [axlVisibleUpdate](#)

**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*.

### **See Also**

[axlVisibleUpdate](#) and [axlVisibleLayer](#)

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

## **axlConductorBottomLayer**

```
axlConductorBottomLayer(  
)  
⇒ t_name
```

### **Description**

Returns the name of the bottom conductor layer.

### **Arguments**

none

### **Value Returned**

*t\_name*      Name of the bottom conductor layer.

### **Example**

```
axlConductorBottomLayer()  
⇒ "BOTTOM"
```

## **axlConductorTopLayer**

```
axlConductorTopLayer()  
⇒ t_name
```

### **Description**

Returns the name of the top conductor layer.

### **Arguments**

none

### **Value Returned**

*t\_name*    Name of the top conductor layer.

### **Example**

```
axlConductorTopLayer()  
⇒ "TOP"
```

## **axIDBCreateFilmRec – Obsolete**

This interface is obsolete. It is kept to support existing SKILL code.

Instead, use [axlFilmCreate](#)

## **axlSetPlaneType**

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

### **Description**

This changes the photoplot type of a conductor or plane type layer between positive or negative artwork. Changing a layer already containing data will require re-voiding existing shapes and updating DRC.

### **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.

### **See Also**

[axlGetParam](#), [axlSetParam](#)

## axlSubclasses

```
axlSubclasses(  
    t_class  
    ?field s_name  
    ?value g_value  
) -> lt_subclasses
```

### Description

Lists subclasses that make up a class. This function is supported in both, APD and Allegro name space. The field and value options provide additional filtering based upon the characteristics of the layer.

The information about the attributes and values permitted on a layer, can be obtained using the following function.

```
axlLayerGet ("MANUFACTURING/PROBE_TOP") ->??
```

You should map the class name via axlMapClassName if you are writing code for both PCB and APD+ as certain class names are different in these products.

The base call is actually just:

```
axlGetParam ("paramLayerGroup:ETCH") ->groupMembers
```

### Arguments

<i>t_class</i>	Class name.
<i>field</i>	Optional field for filtering. If value option is not present filters on basis of a non-nil value.
<i>value</i>	Optional value of field to use for filtering. Requires field option to be passed.

### Values Returned

<i>lt_subclasses</i>	A list of subclass strings.
----------------------	-----------------------------

### See Also

[axlSubclassRoute](#), [axlGetParam](#), [axlMapClassName](#), [axlClasses](#)

## Example

- get all subclasses on class

```
axlSubclasses( axlMapClassName ("MANUFACTURING") )
```

- get user defined subclasses

```
axlSubclasses ("MANUFACTURING" ?field 'userDefined)
```

- all allegro defined

```
axlSubclasses ("MANUFACTURING" ?field 'userDefined ?value nil)
```

## axlSubclassRoute

```
axlSubclassRoute(  
    ?field s_name  
    ?value g_value  
) -> lt_subclasses
```

### Description

Lists subclasses that make up class ETCH.

If no arguments are passed to the function, it returns a list of subclasses in the ETCH class, or CONDUCTOR class, in case of non-PCB product.

The field and value options provide additional filtering based upon the characteristics of the layer. For information on layer parameters, see the section Layer Parameter Attributes (Allegro Subclasses).

The information about the attributes and values permitted on a layer can be obtained using the following command.

```
axlLayerGet( "ETCH/TOP" ) ->??
```

The base call is actually just:

```
axlGetParam("paramLayerGroup:ETCH") ->groupMembers
```

### Arguments

- field**      Optional field for filtering. Uses the value specified by the **value** argument to filter subclasses.
- value**      Optional value of field to use for filtering. Requires **field** option to be passed.

### Values Returned

List of subclass as strings.

### See Also

[axlSubclasses](#), [axlGetParam](#)

## Example

- all etch subclasses

```
axlSubclassRoute() -> ("TOP" "GND" "VCC" "BOTTOM")
```

- all subclasses that are of type etch

```
axlSubclassRoute(?field 'isEtch)
```

- all subclasses that are not etch (e.g dielectric)

```
axlSubclassRoute(?field 'isEtch ?value nil)
```

- all subclasses with material FR-4

```
axlSubclassRoute(?field 'material ?value "FR-4")
```

## **axIXSectionAssign**

```
axIXSectionAssign(  
    t_crossSection/nil  
    g_layer/og_layer  
) -> t/nil
```

### **Description**

Assigns an existing layer to a cross-section. In the recommended model, it is used in building additional cross-sections from the primary cross-section.

Use [axIXSectionCreate](#) to a create a cross-section layer.

This API should be used to manage multiple cross-sections. Should not be used if you are not doing rigid-flex design. It is not supported in the symbol editor.

### **Arguments**

<i>t_crossSection</i>	Name of cross-section  <i>nil</i> is the primary cross-section
<i>g_layer</i>	Can be: <ul style="list-style-type: none"><li>■ <i>o_xsectionDBid</i> - Cross-section dbid</li><li>■ <i>t_layerName</i> - name of etch layer</li><li>■ <i>x_number</i> - position of entry in "All Stackups"</li></ul>
<i>og_layer</i>	List of layers to assign

### **Value Returned**

<i>t</i>	Layer assigned (if layer is already part stackup returns <i>t</i> )
<i>nil</i>	Failed, illegal name, editor does not support this interface or assignment not permitted

### **See Also**

[axIXSectionCreate](#)

### **Examples**

see <cdsroot>/share/pcb/examples/skill/dbcreate/xsection.il

## **axlXSectionCopy**

```
axlXSectionCopy(  
    o_xsectionDBID  
) => g_xsectionDefstruct/nil
```

### **Description**

This copies dbid xsection to an xsection SKILL defstruct. It can be used to duplicate the characteristics of an existing xsection layer to a new layer for use with axlXSectionCreate. Alternatively, it can be used to modify an existing layer via axlXSectionSet, although axlXSectionModify might be a better fit for changing a layer.

This creates a new defstruct using `make_axlXSection()`. The name attribute is never copied.



***If using this to copy named layers, you need to set the name attribute in the defstruct returned by this function.***

### **Arguments**

`o_xsectionDBID`      a xsection dbid

### **Value Returned**

`g_xsectionDefstru`    a defstruct with its attribute data copied from dbid  
`ct`

### **Examples**

- See `<cdsroot>/share pcb/examples/skill/dbcreate/xsection.il`

### **See Also**

[axlXSectionCreate](#), [axlXSectionModify](#)

## **axlXSectionCreate**

```
axlXSectionCreate(
    t_stackup/nil
    g_option
    [g_xsectionDefStruct]
) -> o_xsectionDBID/nil
```

### **Description**

Creates a new cross-section entry.

The three critical create items in `g_xsectionDefStruct` are:

- name – a string which is a name of the layer
- layerType – a string (see `axlXSectionLayerTypes()` for supported types) that describes the type of layer.

Popular types are CONDUCTOR, PLANE, DIELECTRIC or MASK. If no value is specified, DIELECTRIC is used as the default value.

- material – a string that describes substance making up the cross-section layer. If no value is specified, FR-4 is used as the default value for dielectrics and COPPER for conductor or plane layers.

If no attributes are provided, an unnamed cross-section entry of type DIELECTRIC, with material FR-4 is created.

You can set other cross-section entries, see `axlXSectionGet` for a description of the other available attributes. If material is provided it will auto-fill additional attributes based upon the material file entry.

For named cross-section layers, it creates a class/subclass on the etch layers, it does not set the color and visibility for those layers. You need to use `axlLayerSet` for those attributes.

Allegro PCB Editor does not allow name layers above TOP or below BOTTOM layers.

## **CREATING AND POPULATING MULTIPLE CROSS-SECTIONS**

Two models are supported for creating multiple cross-section; assign and create directly. The recommend model is assign.

***The ASSIGN model:***

- Create all the stackups ([axIXSectionCreateStackup](#)).
- Optionally, rename the Primary stackup ([axIXSectionRename](#)).
- Populate the "All Stackups" (use nil as the first argument to [axIXSectionCreate](#)) with the layers. All internal (etch and dielectric) layers are assigned to all stackups.
- Create required mask layers ([axIXSectionCreate](#)).
- Assign ([axIXSectionAssign](#)) the non-mask layers to the non-primary stackups
- Remove ([axIXSectionRemove](#)) any mask layer which is not required in the primary stackup.

By default, all layers created in this model are initially assigned to the primary stackup. The downside with this model is that unwanted layers from the primary stackup are to be removed.

***The CREATE DIRECTLY mode***

- Create all the stackups ([axIXSectionCreateStackup](#)).
- Optionally, rename the primary stackup ([axIXSectionRename](#)).
- Provide a stackup name to [axIXSectionCreate](#) when creating a layer. If the layer is an ETCH layer, it is also assigned to the primary stackup.
- Must assign dielectric layers to primary since only ETCH layers are automatically assigned to this stackup.
- Assign ([axIXSectionAssign](#)) layers to any additional stackups.

The downside of this model is to assign ([axIXSectionAssign](#)) mask or internal layers if required in multiple stackups. This is also true for any dielectric layers that must be in the primary stackup.

**PROGRAMMING TIPS**

- For populating multiple internal layers, use the '`bottom`' option and build the stackup from bottom to top.
- Use '`afterBottom`' option to assign MASK layers below BOTTOM.
- Ensure that there is at least one dielectric layer between each CONDUCTOR or PLANE layer type. Adjacent routing layers are allowed, but this results in incorrect signal analysis.

## Allegro SKILL Reference

### Parameter Management Functions

---

- Do not name dielectric layers unless you want to physically add and edit objects to those layers. In Allegro PCB Editor, every named layers creates an entry in the class/subclass table on ETCH related classes.
- Material names that are not contained in the materials file, can also be specified. Setting environment variable “`xsection_material_warning`” prints a warning. When unknown materials are specified the thickness and other associated data are not populated.
- Changing the material also updates its associated data. This is only an issue with the `axlXSectionSet` API.
- `layerId` defaults to the layer number.
- When creating MASK layers, you must provide the `?name` option. If you provide both, a `?name` and `?maskLayer` option, then `?maskLayer` name wins.
- MASK layers added between TOP and BOTTOM do not effect connectivity.
- Constraint Manager removes the ‘\_’ in pretty printing `layerFunction`. This interface requires them (see types returned by `axlXSectionLayerFunctions` API). Mask and dielectric have different allowed function types.
- Use `axlXSectionModify` or `axlXSectionCopy` to create the `axlXSection` defstruct. Do not directly call `make_axlXSection`.
- DRC is marked out of date with cross-section changes.

## Allegro SKILL Reference

### Parameter Management Functions

---

#### Arguments

<i>nil</i>	default stackup (primary or All stackups)
<i>t_stackup</i>	in a multi-stackup design the name of the stackup
<i>g_option</i>	
	' <i>top</i> insert layer above TOP. For PCB designs this can only be a unnamed dielectric or MASK layers.
	' <i>bottom</i> insert layer above BOTTOM
	' <i>afterBottom</i> insert layer after BOTTOM. For PCB Designs this can only be unnamed dielectrics or MASK layers.
<i>t_etchSubclass</i>	subclass name, insert layer above this name
<i>x_position</i>	insert above cross-section position. This is the number field in the cross-section dbid. In designs with one cross-section, you can also use the position attribute. You cannot insert above 0 (Surface).
<i>g_xsectionDefStruct</i> or <i>nil</i>	A SKILL defstruct with all possible attributes for cross-section entry
	<i>make_axlXSection</i> creates a new entry
	<i>copy_axlXSection</i> copies an existing entry
	<i>axlXSectionCopy</i> copies contents of an existing cross-section dbid to a new defstruct.

#### Value Returns

*o\_xsectionDBID* if successful

nil                    failed.

## Examples

See <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il

## See Also

[axIXSectionGet](#), [axIXSectionSet](#), [axIXSectionDelete](#), [axIXSectionLayerTypes](#),  
[axIXSectionCopy](#), [axIXSectionModify](#), [axIXSectionLayerFunctions](#),  
[axIXSectionCreateStackup](#), [axIXSectionDeleteStackup](#), [axIXSectionRename](#),  
[axIXSectionAssign](#), [axIXSectionRemove](#), [axIZoneCreate](#)

## **axlXSectionCreateStackup**

```
axlXSectionCreateStackup(  
    t_crossSection  
) -> t/nil
```

### **Description**

This creates a new empty stackup with name, *t\_name*.

Should not be used if you are not doing rigid-flex design. It is not supported in the symbol editor.

### **Arguments**

*t\_crossSection*      Name of stackup (case insensitive)

### **Value Returned**

*t*                    Stackup created

*nil*                  Failed, stackup exists, illegal name, editor does not support this interface or exhausted the maximum number of stackups

### **See Also**

[axlXSectionCreate](#)

### **Examples**

See <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il

## **axlXSectionDelete**

```
axlXSectionDelete(  
    g_option  
) => lt_types
```

### **Description**

Deletes a cross section layer. If layer is associated with an ETCH layer, the associated layer is also deleted. Associated ETCH layers must not have objects. See [axlLayerDelete](#) for other reasons for failure.

### **Arguments**

*g\_option*  
*o\_xsectionDBID* delete layer by XSection dbid  
*t\_etchSubclass* delete layer by this name  
*x\_position* delete layer by position. This is the position attribute in the xsection dbid.

### **Value Returned**

*t* if successful  
*nil* failed.

### **Examples**

See <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il

### **See Also**

[axlXSectionCreate](#), [axlLayerCreate](#), [axlDeleteByLayer](#)

## **axlXSectionDeleteStackup**

```
axlXSectionDeleteStackup(  
    t_crossSection  
) -> t/nil
```

### **Description**

Deletes a stackup with name `t_name`. Cannot delete a stackup assigned to a zone.

### **Arguments**

`t_crossSection`      Name of stackup (case insensitive)

### **Value Returned**

`t`                    Stackup is deleted

`nil`                  Stackup is in use or not found

### **See Also**

[axlXSectionCreate](#)

### **Examples**

Delete a "flex1" created by RigidFlexCreate()

```
in <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il
```

see [axlXSectionCreateStackup](#)

```
axlXSectionDeleteStackup("flex1")
```

## **axlXSectionGet**

```
axlXSectionGet(  
    g_stackup/nil  
    g_option  
)  
==> g_data/nil
```

### **Description**

Returns data about the cross-section entry for a design.

Supports multiple access options (see Arguments).

Allegro design color and visibility query needs are satisfied by the [axlLayerGet](#) API.

## Allegro SKILL Reference

### Parameter Management Functions

---

#### Arguments

<i>g_stackup</i>		
<i>nil</i>	for cross-sections with a single stackup this returns the standard stackup. For Rigid-Flex designs this returns All Stackups.	
<i>str</i>	name of stackup. From zone groups it is the name in zoneDbid -> stackup or available stackups are returned by axlXSectionGet(nil 'stackups)	
<i>g_option</i>		
' <i>stackups</i>	return a list of stacks. g_xsection is ignored and should be nil	
' <i>count</i>	return number of cross-section entries for provided stackup. g_data is x_entries	
' <i>number</i>	return maximum number of individual stackups database supports	
' <i>thickness</i>	return provided stackup thickness in user units g_data is f_thickness design units, accuracy is not restricted to current design accuracy. This is the total thickness with masks.	
' <i>top</i>	return cross-section entry for top layer of design	
' <i>bottom</i>	return cross-section entry for bottom layer of design	
<i>x_position</i>	return cross-section entry for given position in "All stackups". This is the number field in a cross-section dbid. (0 is the 1st layer -- air)	
<i>t_etchSubclas</i> <i>s</i>	name of etch subclass. g_data is a o_xsectionDBID	
' <i>all</i>	return entire stackup. g_data must be a lo_xsectionDBID	

## Allegro SKILL Reference

### Parameter Management Functions

---

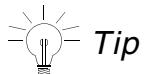
'locked

lock status

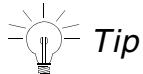
- nil - not locked

list of lock status. This is a user interface lock only.  
SKILL and techfile can make changes.

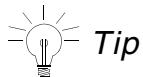
- 'value: cannot edit values
- 'layer: add or delete layers



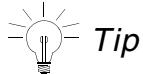
The name of the stackup can be obtained from the zone dbid by zoneDbid->stackup



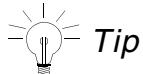
If using a single stackup in a design pass nil as the first argument.



If a multi-stackup design, a nil returns all of the cross-section dbids (for example, "All Stackups"). If you need the main stackup, typically called "primary", use its name.



Except for 'stackups and 'locked all other options take into account the stackup argument.



For 'thickness if g\_xsection is nil, the stackup thickness is returned, which can be overridden by the TEXT\_BOARD\_THICKNESS property assigned at the design level. Otherwise, if a stackup (g\_xsection) name is provided, the calculated thickness of the indicated stackup is returned.

## Allegro SKILL Reference

### Parameter Management Functions

---

#### **XSection Attributes:**

Attributes that have Yes under Modify column can be set with APIs [axIXSectionCreate](#) and [axIXSectionSet](#).

NAME	TYPE	Modify	DESCRIPTION
objType	string	No	"xsection"
readOnly	t	No	Cannot directly modify entry
constraint	string	Yes	(Optional) Techfile generic layer support (max 1023 chars).
conductor	t/nil	Yes	Is layer of conductor or plane?
layerType	string	Yes	Type of xsection layer.
layerFunction	string	Yes	Layer function of a layer. This is a superset of layer types for dielectric and mask layers. Conductor, plane and surface layers cannot have their layerFunction changed. When creating a layer, Allegro automatically assigns a default function based on the layer type. Can be nil (surface layers).
layerId	string	Yes	Override of via label id (max 3 characters)
material	string	Yes	Layer material (max 250 chars)
maskLayer	string	Yes	Mask layer associated with this cross-section entry.
			This is nil unless the layerType is type MASK. If a string is present it will have the same name as the name field.
mfg	string	Yes	(Optional) Stackup layer grouping for IPC2581 (max 255 chars)
name	string/nil	Yes	Name of xsection/etch layer (max 47 chars).

## Allegro SKILL Reference

### Parameter Management Functions

---

<b>NAME</b>	<b>TYPE</b>	<b>Modify</b>	<b>DESCRIPTION</b>
negativeArtwork	t/nil	Yes	Layout is a negative image.
polyCutLayer	t/nil	Yes	Layer is a cut layer for poly vias. Only dielectric layers between the surface layers may be cut layers. You cannot unset a cut layer that is being used by a poly via.
position	integer	No	Position of xsection layer in stackup. This is relative to the stackup. (starts at 0 for SURFACE).
number	integer	No	Absolute position which is the position in "All Stackups". In non-multi cross-section designs this has the same value as position.
prop	l_dbid	Yes	List of properties on object.
thickness	float	Yes	layer thickness in design units, stored with more accuracy than design
tolMinus	float	Yes	thickness tolerance - in design units, stored with more accuracy than design
tolPlus	float	Yes	thickness tolerance + in design units, stored with more accuracy than design
Pad suppression:			
unusedPin	t/nil	Yes	suppress unused pads of pins on this layer
unusedVia	t/nil	Yes	suppress unused pads of vias on this layer
Embedded support:			

## Allegro SKILL Reference

### Parameter Management Functions

---

<b>NAME</b>	<b>TYPE</b>	<b>Modify</b>	<b>DESCRIPTION</b>
embedded	string	Yes	Type of embedded layer (default NOT_EMBEDDED)  Values: nil, NOT_EMBEDDED, BODY_UP, BODY_DOWN PROTRUDING_ALLOWED
embeddedAttach	string	Yes	Type of attachment if embedded layer.  Values: nil, DIRECT_ATTACH, INDIRECT_ATTACH
Electrical parameters:			
diffCouplingType	string	Yes	Diffpair routing on layer (Values: nil or EDGE)
diffSpacing	float	Yes	Typical diffpair spacing on this layer in design units (only if coupling type is set)
conductivity	string	Yes	Layer conductivity (MKS in mho/cm)
dielectricConst	string	Yes	Dielectric constant
etchFactor	integer	Yes	Vertical geometry of etch (angle 45-135 or 225-315 degrees)
freqDepFile	string	Yes	(optional) Frequency dependent file for this layer (max 255 chars)
lossTangent	string	Yes	Loss tangent value
shield	t/nil	Yes	Is this a shield layer
width	float	Yes	Typical layer width for Impedance what-ifs in design units
siIgnore	t/nil	Yes	SI should ignore this layer

## Value Returned

nil	an error
g_data	depends on g_option, see <b>Arguments</b> section

## Examples

- Get design thickness

```
thick = axlXSectionGet(nil 'thickness)
```

- Fetch entire cross-section

```
stackup = axlXSectionGet(nil 'all)
```

- Fetch just the TOP layer

```
xs = axlXSectionGet(nil 'top)
```

- Fetch dielectric below top-most layer

```
xs = axlXSectionGet(nil 'top)
dielectric = axlXSectionGet(nil xs->number +1)
axlPrintDbid(dielectric)
```

- Get all stackups (even with one stackup will return PRIMARY)

```
stackups = axlXSectionGet(nil 'stackups)
```

- Get stackup called FLEX2

```
zoneStackup = axlXSectionGet("FLEX2" 'all)
```

- Get number of layers in FLEX2 (includes etc, dielectric and layers above surface)

```
zoneStackup = axlXSectionGet("FLEX2" 'count)
```

- See <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il

## **axlXSectionLayerFunctions**

```
axlXSectionLayerFunctions()  
    => lt_types
```

### **Description**

Return list of supported layer function types. Layer functions are a super set of layer types.

### **Arguments**

None

### **Value Returned**

- List of strings of supported layer types.

### **Examples**

Get the list of layer functions:

```
types = axlXSectionLayerFunctions()
```

### **See Also**

[axlXSectionCreate](#)

## **axlXSectionLayerTypes**

```
axlXSectionLayerTypes()  
    => lt_types
```

### **Description**

Return list of supported layer types. This is used with layerType attribute when adding or modifying xsection layers.

### **Arguments**

None

### **Value Returned**

- List of strings of supported layer types.

### **See Also**

[axlXSectionCreate](#)

### **Examples**

```
only use types = axlXSectionLayerTypes()
```

## **axlXSectionModify**

```
axlXSectionModify(  
    <list of defstruct attributes and values>  
) => g_xsectionDefstruct/nil
```

### **Description**

This is a wrapper for `make_axlXSection()` function to create a new defstruct. For certain fields (for example, boolean and some string fields) a '`none` symbol is used to indicate that the field should not be changed, other fields can use the default `nil` symbol.

### **Arguments**

Same as `make_axlXSection()`.

### **Value Returned**

`g_xsectionDefstruc` a defstruct with its attribute data copied from provided  
`ct` arguments

### **Examples**

- Modify a layer to be negative artwork assume a layer named GND exists. All other layer characteristics will remain the same

```
xs = axlXSectionGet(nil "GND")
negative = axlXSectionModify(?negativeArtwork t)
ret = axlXSectionSet(xs negative)
```

- See `<cdsroot>/share pcb/examples/skill/dbcreate/xsection.il`

### **See Also**

[axlXSectionCreate](#)

## **axIXSectionRemove**

```
axIXSectionRemove(  
    t_crossSection/nil  
    g_layer/og_layer  
) -> t/nil
```

### **Description**

Removes a layer from indicated stackup. The layer is not deleted, it is de-assigned from a stackup. There are rules (see cross-section documentation) for layer membership in stackups. If you do something not permitted a nil is returned.

Its typical use is when building a multi-cross section design. It removes mask layers from the primary cross-section.

This is used to manage multiple cross-sections. It is not applicable to a single cross-section design. It is not supported in the symbol editor.

Use [axIXSectionDelete](#) if a layer should be deleted.

### **Arguments**

<i>t_name</i>	Name of stackup (case insensitive) if nil, it uses the Primary stackup
<i>g_layer</i>	Can be: <ul style="list-style-type: none"><li>■ o_xsectionDBid: Cross-section dbid</li><li>■ t_layerName: Name of etch layer</li><li>■ x_number: Position of entry in "All Stackups"</li></ul>
<i>og_layer</i>	List of layers to remove

### **Value Returned**

<i>t</i>	Layer removed from stackup (if layer is already not part of the stackup it returns <i>t</i> )
nil	Failed, illegal name, editor does support interface, layer cannot be removed

### **See Also**

[axlXSectionCreate](#)

**Examples**

See <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il

## **axlXSectionRename**

```
axlXSectionRename(  
    t_crossSection/nil  
    t_newName  
) -> t/nil
```

### **Description**

Renames a stackup. If first argument is nil, renames the primary stackup.

Should not be used if you are not doing rigid-flex design. It is not supported in the symbol editor.

### **Arguments**

<i>t_crossSection</i>	Existing name of stackup
<i>t_newName</i>	New name of stackup (case insensitive)

### **Value Returned**

<i>t</i>	Stackup renamed
<i>nil</i>	Failed

### **See Also**

[axlXSectionCreate](#)

### **Examples**

Assume a multi-stackup design rename the default primary name

```
axlXSectionRename(nil "rigid")
```

## **axIXSectionSet**

```
axIXSectionSet(
    g_option
    [g_xsectionDefStruct]
) -> t/nil
```

```
axIXSectionSet(
    'locked
    nil/l_lockType
) -> t/nil
```

### **Description**

Modifies an existing cross-section entry. Use [axIXSectionModify](#) to create a xsection defstruct with the attributes you wish to modify.

See [axIXSectionGet](#) for a description of the other available attributes. If changing material, it also updates the material characteristics to values of the new material.

Allegro PCB editor does not allow name layers above top or below bottom.

### **PROGRAMM TIPS:**

- ❑ See tips in [axIXSectionCreate](#).
- ❑ In Allegro PCB Editor, you cannot rename TOP or BOTTOM.
- ❑ Cannot unname a layer with data on the ETCH layers.
- ❑ DRC is marked out of date.

## Allegro SKILL Reference

### Parameter Management Functions

---

#### Arguments

<i>nil</i>	first argument must always be <i>nil</i>
<i>g_option</i>	
<i>o_xsectionDBID</i>	delete layer by cross-section dbid
<i>t_etchSubclass</i>	modify by subclass name
<i>x_position</i>	modify by cross-section position. This is the number attribute in the cross-section dbid.
' <i>locked</i>	update user interface layer locking
<i>g_xsectionDefStruct</i>	A SKILL defstruct with all possible attributes for cross- section entry
<i>make_axlXSection</i>	creates a new entry
<i>copy_axlXSection</i>	copies an existing entry
<i>axlXSectionCopy</i>	copies contents of an existing cross-section dbid to a new defstruct.
<i>l_lockType</i>	change user interface locking
	<ul style="list-style-type: none"><li>■ <i>nil</i> - unlock</li><li>■ '<i>layer</i> - prevent layer addition or deletion</li><li>■ '<i>value</i> - prevent changing values</li><li>■ list of both '<i>layer</i> and '<i>value</i></li></ul>

#### Value Returned

<i>t</i>	if successful
<i>nil</i>	failed.

#### See Also

[axlXSectionCreate](#), [axlXSectionCopy](#)

## Examples

■ See <cdsroot>/share pcb/examples/skill/dbcreate/xsection.il

■ lock layer values

```
axlXSectionSet('locked 'layer)
```

---

# 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 Allegro 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 Allegro 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 Allegro PCB Editor Database User Model.”](#) for a list of Allegro 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 Allegro 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 Allegro 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 Allegro PCB Editor object types by name:

- |           |  |
|-----------|--|
| NET       | List of net names for net selection.                               |
| COMPONENT | List of reference designators for component instance selection.    |
| SYMBOL    | List of reference designators for symbol instance selection.       |
| FUNCTION  | List of function designators for function instance selection.      |
| DEVTYPE   | List of device type for component or symbol instance selection. ** |

## Allegro SKILL Reference

### Selection and Find Functions

---

SYMTYPE	List of symbol types for symbol instance selection.
REFDES	List of reference designators for component or symbol instance selection. **
DEVSYM	List of device type for symbol instance selection.
GROUP	List of group names for group or group member selection.
PROPERTY	List of property names or property value) lists for selection of database object that have the property/value. If no value is provided, the value will be ignored for the database property comparison. The find filter enabled and onButtons control the type of elements that will be selection.

\*\* For DEVTYPE and REFDES, a component instance is selected if COMPONENTS are in the find filter ?enabled list. Otherwise, if SYMBOLS are enabled, a symbol instance is selected.

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 Allegro 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 Allegro PCB Editor Find Filter. You can prefix any of these keywords with `NO` to reverse the effect. See the description of [axlSetFindFilter](#) on page 312 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.
	<b>Note:</b> 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]  
    [trapEnable]  
)  
⇒ t/nil
```

#### **Description**

Finds an object that is close to the input point and meets the *Find* filter criteria and adds that object to the selection set after clearing all previous selections. If no point is provided, prompts to select a point prior to the find operation.

`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 279.)

This finds objects within the current trapsize (`axlGetTrapBox`), which varies based upon the zoom factor.

## Arguments

<i>l_point</i>	Coordinate pair for xy location where objects are to be found.
<i>trapEnable</i>	By default, enables a trap box, which is based on the zoom level of the canvas. Objects that are found can be effected based on the current zoom level. When this option is set to <code>nil</code> the trap size gets disabled.

## Value Returned

<code>t</code>	One or more <i>dbids</i> put in the select set.
<code>nil</code>	No <i>dbids</i> put in the select set.

**Note:** An object which is currently in the selection set may be reselected if there are no unselected objects that qualify for selection at the given point and find filter.

## Example

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

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

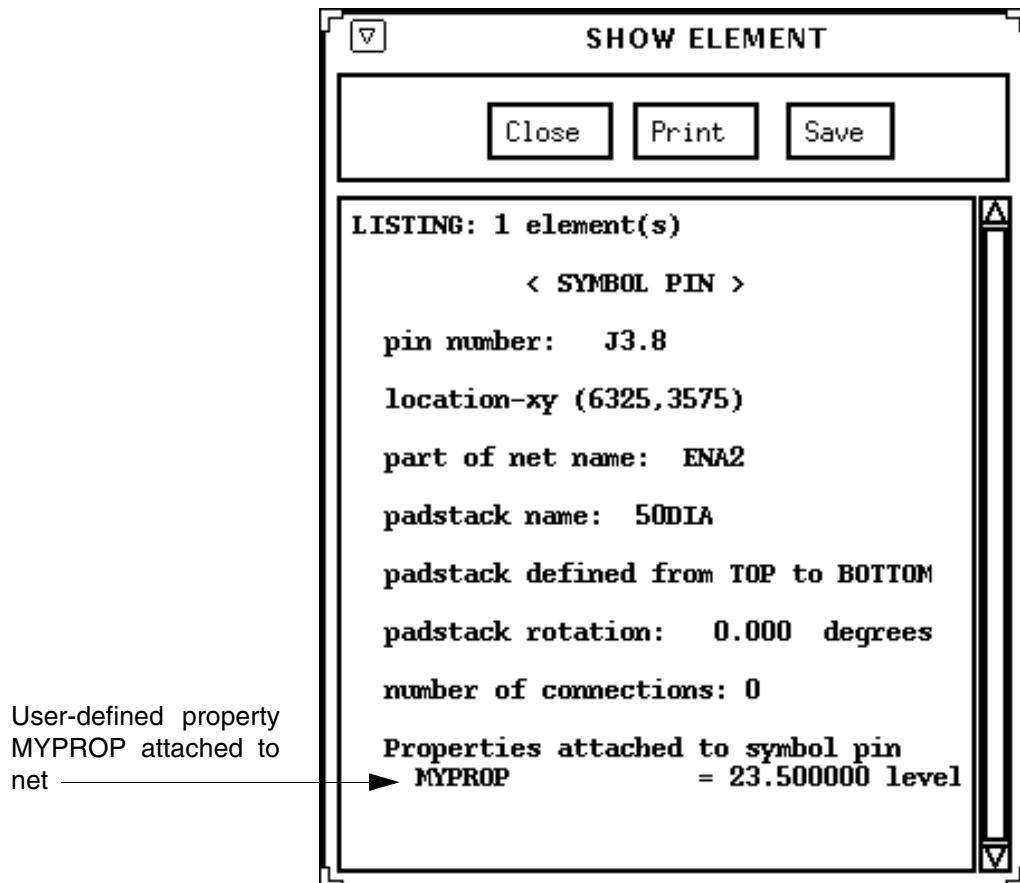
To check

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

## Allegro SKILL Reference

### Selection and Find Functions

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



### See Also

[axlAddSelectPoint](#)

## axlAddSelectPoint

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

### Description

Finds a non-selected object that is close to the input point and meets the *Find* filter criteria and adds that object to the selection set. If no point is provided, prompts to select a point prior to the find operation.

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. Adds all the qualified figures that belong to the net to the select set. (See the [Find Filter Options](#) on page 279.)

This function finds objects within the current trapsize (`axlGetTrapBox`), which varies based upon the zoom factor.

### Arguments

<i>l_point</i>	Coordinate pair for xy location where objects are to be found.
<i>trapEnable</i>	Set to <code>nil</code> to restrict found elements to exactly matching location, instead of close to the location.

### Value Returned

<i>t</i>	One or more <code>dbids</code> put in the select set.
<i>nil</i>	No <code>dbids</code> put in the select set.

### Example

See [axlSingleSelectPoint](#) on page 280 for an example. `axlSingleSelectPoint` has the same behavior except that it selects only one object.

## **Allegro SKILL Reference**

### Selection and Find Functions

---

#### **See Also**

[axlSingleSelectPoint](#), [axlSubSelectBox](#), [axlSingleSelectBox](#), [axlAddSelectBox](#),  
[axlSubSelectPoint](#)

## axlSubSelectPoint

```
axlSubSelectPoint(  
    [l_point]  
    [trapEnable]  
)  
⇒ t/nil
```

### Description

Finds a selected object that is close to the input point and meets the find filter criteria and removes that object from the selection set. If no point is provided, prompts to select a point prior to the find operation.

Removes a *single* object from the select set, unless EQUIVLOGIC is on. In that case, it may find 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 279.)

This function finds objects within the current trapsize (axlGetTrapBox), which varies based upon the zoom factor.

### Arguments

*l\_point* Coordinate pair for xy location where objects are to be found.

*trapEnable* Set to *nil* to restrict found elements to exactly matching location, instead of close to the location.

### Value Returned

*t* One or more *dbids* removed from the select set

*nil* No *dbids* removed from the select set.

### Example

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 in the following example:

## Allegro SKILL Reference

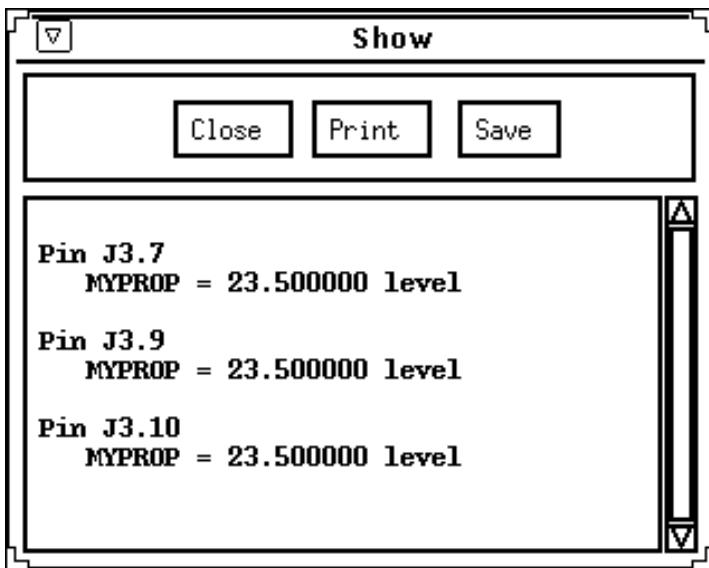
### Selection and Find Functions

```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")
axlSingleSelectBox( list(6200:3700 6500:3300))
axlSubSelectPoint( 6325:3575 )
axlDBAddProp(axlGetSelSet() list("MYPROP" 23.5))
⇒ t
```

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*.



### See Also

[axlAddSelectPoint](#)

## **axlSingleSelectBox**

```
axlSingleSelectBox(  
    [l_bBox]  
    [trapEnable]  
)  
⇒ t/nil
```

### **Description**

Finds all objects that intersect with the input box and meet the find filter criteria and adds those objects to the selection set after clearing all previous selections. If no box (or an incomplete box) is provided, prompts to select a box (or to complete the box) prior to the find operation.

### **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 <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.
<i>trapEnable</i>	Set to <i>nil</i> to restrict found elements to exactly matching location, instead of close to the location.

### **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**

Adds a previously defined user property `MYPROP` to four pins by selecting a box around them with corners (6200:3700 6500:3300).

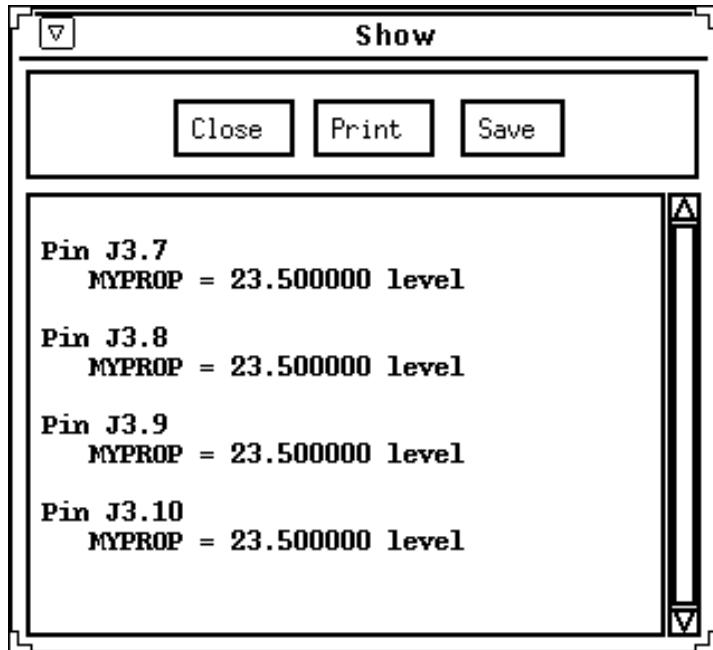
```
axlSetFindFilter (?enabled "pins" ?onButtons "pins")  
axlSingleSelectBox( list(6200:3700 6500:3300))  
axlDBAddProp(axlGetSelSet() list( "MYPROP" 23.5))  
⇒ t
```

To check

1. Select *Edit – Properties*.

2. Select the four pins.
3. Select *MYPROP* from the Available Properties list in the Edit Property window.

The command displays the four pins in the Show Properties window as having *MYPROP* with value *23.500000 level*



## See Also

[axlAddSelectPoint](#), [axlAddSelectBox](#)

## **axlAddSelectBox**

```
axlAddSelectBox(  
    [l_bBox]  
    [trapEnable]  
)  
⇒ t/nil
```

### **Description**

Finds all non-selected objects that intersect with the input box and meet the find filter criteria and adds those objects to the selection set. If no box (or an incomplete box) is provided, prompts to select a box (or to complete the box) prior to the find operation.

### **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.
- trapEnable* Set to `nil` to restrict found elements to exactly matching location, instead of close to the location.

### **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 287. That function behaves exactly as `axlAddSelectBox`, except that `axlSingleSelectBox` does not clear the select set.

### **See Also**

[axlAddSelectPoint](#), [axlSingleSelectBox](#)

## **axlSubSelectBox**

```
axlSubSelectBox(  
    [l_bBox]  
    [trapEnable]  
)  
⇒ t/nil
```

### **Description**

Finds all selected objects that intersect with the input box and meet the find filter criteria and removes those objects from the selection set. If no box (or an incomplete box) is provided, prompts to select a box (or to complete the box) prior to the find operation.

### **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.
<i>trapEnable</i>	Set to <i>nil</i> to restrict found elements to exactly matching location, instead of close to the location.

### **Value Returned**

<i>t</i>	One or more objects deleted from select set.
<i>nil</i>	No objects deleted from select set.

### **Example**

See [axlSubSelectPoint](#) on page 285 for an example of subtracting from the select set, and [axlAddSelectPoint](#) on page 283 for an example of using a box for selection.

### **See Also**

[axlAddSelectPoint](#)

## **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.

**Note:** Dependent on find filter ?enabled settings.

### Example 1

The following example selects the nets GND and VCC by their names.

```
axlClearSelSet()  
axlSetFindFilter(?enabled  
    list( "noall" "equivlogic" "nets" "clines" "vias")  
    ?onButtons list( "all"))  
axlSingleSelectName( "NET" list( "GND" "VCC"))  
    ==> (dbid:234436 dbid:98723)  
axlSingleSelectName("PROPERTY" list( list("BUS_NAME" "MEM") "FIXED"  
)⇒ 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.

## Example 2

The following example selects a set of nets--one set by the property name ELECTRICAL\_CONSTRAINT\_SET with value DEFAULT, and another set that has the name ROUTE\_PRIORITY.

```
axlClearSelSet()  
axlSetFindFilter( ?enabled list( "noall" "nets")  
                  ?onButtons list( "all"))  
axlSingleSelectName("PROPERTY"  
                    list( list( "ELECTRICAL_CONSTRAINT_SET" "DEFAULT"  
                               "ROUTE_PRIORITY"))  
==> (dbid:234436 dbid:98723 dbid:234437 dbid:98727  
      dbid:234438 dbid:98725 dbid:234439 dbid:98726)
```

## **axlSingleSelectName**

```
axlSingleSelectName (
  t_nameType
  l_names
  [g_wildcard]
)
⇒ 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. Both `axlSubSelectName` and `axlAddSelectName` always operate in wildcard mode. Both take the optional wildcard argument but ignore it. In the future, if passed `nil`, they may obey the argument.

### **Arguments**

<i>t_nameType</i>	Indicates the type of name string being provided. Also implies the type of object to be selected. (see <a href="#">Finding Objects by Name</a> on page 276).
<i>l_names</i>	List of item name names or for properties a name value. This varies with the nameType. Supports a single name item outside of a list. Therefore, supported values can be: <ul style="list-style-type: none"><li>– a name</li><li>– a list of names</li><li>– a list of name/value pairs</li></ul>
<i>g_wildcard</i>	A <i>t</i> means that * or ? performs regular expression matching. Default is <code>nil</code> where * and ? are treated as normal characters.

### **Value Returned**

- t* One or more objects added to the select set.
- nil* 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 <a href="#">Finding Objects by Name</a> on page 276). Also implies the type of object to be selected.
<i>l_names</i>	List of item name names or for properties a name value. This varies with the nameType. Supports a single name item outside of a list. Therefore, supported values can be: <ul style="list-style-type: none"><li>– a name</li><li>– a list of names</li><li>– a list of name/value pairs</li></ul>

### Value Returned

<i>t</i>	One or more objects added to the select set.
<i>nil</i>	No objects added to the select set.

The *on* buttons matter for select Properties by name, but don't for other name types. Both `axlSubSelectName` and `axlAddSelectName` always operate in wildcard mode. Both take the optional wildcard argument, but ignore it. In the future, if passed `nil` they may obey the argument.

## Example

See examples for [axlSingleSelectName](#) on page 294. 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 <a href="#">Finding Objects by Name</a> on page 276). Also implies the type of object to be selected.
<i>l_names</i>	List of item name names or for properties a name value. This varies with the nameType. Supports a single name item outside of a list. Therefore, supported values can be: <ul style="list-style-type: none"><li>– a name</li><li>– a list of names</li><li>– a list of name/value pairs</li></ul>

### Value Returned

<i>t</i>	One or more objects deleted from select set.
<i>nil</i>	No objects deleted from select set.

**Note:** The *on* buttons matter for select Properties by name, but not for other name types. Both `axlSubSelectName` and `axlAddSelectName` always operate in wildcard mode. Both take the optional wildcard argument, but ignore it. In the future, if passed `nil`, they may obey the argument.

## Example

See examples for [axlSingleSelectName](#) on page 294. 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.

**Note:** You need to "BOUNDARY\_SHAPES" to the axlSetFindFilter "?enabled" option if you wish to select a dynamic shape. Generated shape from dynamics shapes cannot be selected via the SelectObject APIs since it cannot be edited.

### **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(  
    ?firstEventCallbacks_callback  
    ?groupModet/nil  
    ?promptt_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 Allegro 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 Allegro 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.
<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.

## **Allegro SKILL Reference**

### Selection and Find Functions

---

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> .
nil	Select set is empty.

### **Example**

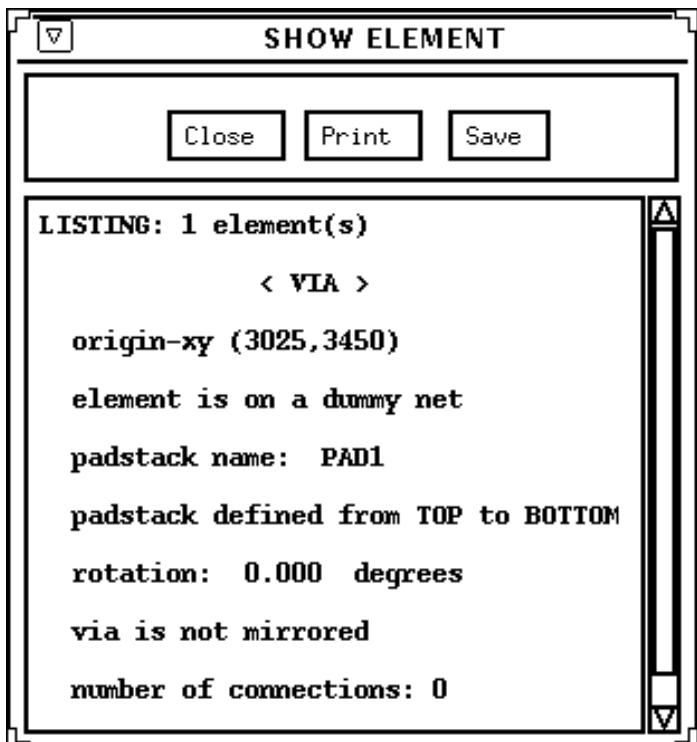
```
axlClearSelSet()  
axlSetFindFilter(?enabled list("noall" "vias")  
                 ?onButtons list("vias"))  
axlAddSelectBox(list(3000:3100 3200:3600))  
axlShowObject(axlGetSelSet())  
⇒ t
```

Selects all vias in a box area and shows the contents of the select set.

## Allegro 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.

### **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(  
    [onEnabledF]  
)  
⇒ lt_filters/nil
```

### Description

Returns the current Find Filter settings as a list of keyword strings. The return find filters settings (onButton or enabled) is controlled by the boolean `onEnabledF`.

Note that SKILL has its own local Find Filter settings. The `axlGetFindFilter()/axlSetFindFilter()` API is for getting and setting the local Find Filter settings. This does not get the global Find Filter settings, which can only be modified by the user. SKILL programs are not allowed to modify the global Find Filter settings.

### Arguments

`onEnabledF`      If `onEnabledF` is `t`, returns the enabled list.  
                        If `onEnabledF` is `nil`, returns the `onButton` list.  
                        Default is `nil`.

### Value Returned

`lt_filters`      List of element types in the Find Filter  
                        OR  
                        list of current onButton settings.  
`nil`                If list would be empty.

### Example

```
axlSetFindFilter ?enabled (list "vias" "pins" "nets" "clinesegs" "nameform")  
?onButtons (list "vias" "pins" "clinesegs"))  
ret =axlGetFindFilter()
```

Returns the following:

```
("NAMEFORM" "NETS" "CLINESEGS" "VIAS" "PINS"  
<plus default items that may change from release to release> )
```

## **Allegro SKILL Reference**

### Selection and Find Functions

---

```
ret = axlGetFindFilter(t)
```

**Returns:**

```
("CLINESEGS" "VIAS" "PINS")
```

## **axlSetFindFilter**

```
axlSetFindFilter(  
    ?enabledlt_enabled  
    ?onButtonslt_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**.

Note that SKILL has its own local Find Filter settings. The axlGetFindFilter()/axlSetFindFilter() API is for getting and setting the local Find Filter settings. This does not get the global Find Filter settings, which can only be modified by the user. SKILL programs are not allowed to modify the global Find Filter settings.

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**

<b>Keyword</b>	<b>Description</b>
"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

**axlSetFindFilter Keywords, continued**

---

<b>Keyword</b>	<b>Description</b>
"DRCS"	Enable DRC errors
"TEXT"	Enable text
"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
"BONDWIRES"	Enable bondwires (APD/SIP only).  For more information, see <a href="#">Bond Objects</a> .
"FINGERS"	Enable fingers (APD/SIP only).  For more information, see <a href="#">Bond Objects</a> .

## **Allegro SKILL Reference**

### Selection and Find Functions

## **ax1SetFindFilter** Keywords, *continued*

Keyword	Description																			
"GROUPS"	"GROUPS" and "GROUPMEMBERS" operate together to produce four possible selection states:																			
	<table border="1" data-bbox="535 536 1153 756"> <thead> <tr> <th data-bbox="535 536 789 606" rowspan="2">Keyword</th><th colspan="4" data-bbox="789 536 1153 563">States</th></tr> <tr> <th data-bbox="820 585 847 606">1</th><th data-bbox="878 585 904 606">2</th><th data-bbox="936 585 962 606">3</th><th data-bbox="995 585 1021 606">4</th></tr> </thead> <tbody> <tr> <td data-bbox="535 642 789 663">GROUPS</td><td data-bbox="820 642 847 663">OFF</td><td data-bbox="878 642 904 663">ON</td><td data-bbox="936 642 962 663">OFF</td><td data-bbox="995 642 1021 663">ON</td></tr> <tr> <td data-bbox="535 696 789 718">GROUPMEMBERS</td><td data-bbox="820 696 847 718">OFF</td><td data-bbox="878 696 904 718">OFF</td><td data-bbox="936 696 962 718">ON</td><td data-bbox="995 696 1021 718">ON</td></tr> </tbody> </table>	Keyword	States				1	2	3	4	GROUPS	OFF	ON	OFF	ON	GROUPMEMBERS	OFF	OFF	ON	ON
Keyword	States																			
	1	2	3	4																
GROUPS	OFF	ON	OFF	ON																
GROUPMEMBERS	OFF	OFF	ON	ON																
	<ul style="list-style-type: none"> <li>■ State 1: Legacy. This supports code that predates the group implementation. This is the same as State 3.</li> </ul>																			
	<ul style="list-style-type: none"> <li>■ State 2: Group only. By only setting the group bitfield, any selected group is returned to the application as a group.</li> </ul>																			
	<ul style="list-style-type: none"> <li>■ State 3: Members only. By only setting the group_members bitfield, group members are returned to the application when a group is selected.</li> </ul>																			
	<ul style="list-style-type: none"> <li>■ 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.</li> </ul>																			
"AUTOFORM"	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 class/subclass form setting.																			
"NAMEFORM"	Option - enables the find by name/property fields in the find filter form.																			

**axlSetFindFilter Keywords, continued**

---

<b>Keyword</b>	<b>Description</b>
"DYNTHEMALS"	<p>Option - Enable selection of thermal 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 from read-only purposes.</p> <p>If in the partition editor or the design has partitions active then this option is used to selections of read-only objects.</p> <p>This option should also be used to select shape base fillets. The system will not allow shape based fillets to be modified if the dynamic fillet option is enabled.</p>
"BONDSMART"	<p>Option - Application has been updated to differentiate bond wires and/or fingers (APD/SIP).</p> <p>For more information, see <a href="#">Bond Objects</a>.</p>
"ALLTYPES"	Enable all object types ("PINS" ... "NETS").
"ALL"	Enable all object types and options.

---

`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. Additional information is useful only if your SKILL program access shapes.

A shape on ETCH may be either static or dynamic. A static shape is similar to what existed in Allegro PCB Editor prior to SPB15.0. 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 auto-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 (for

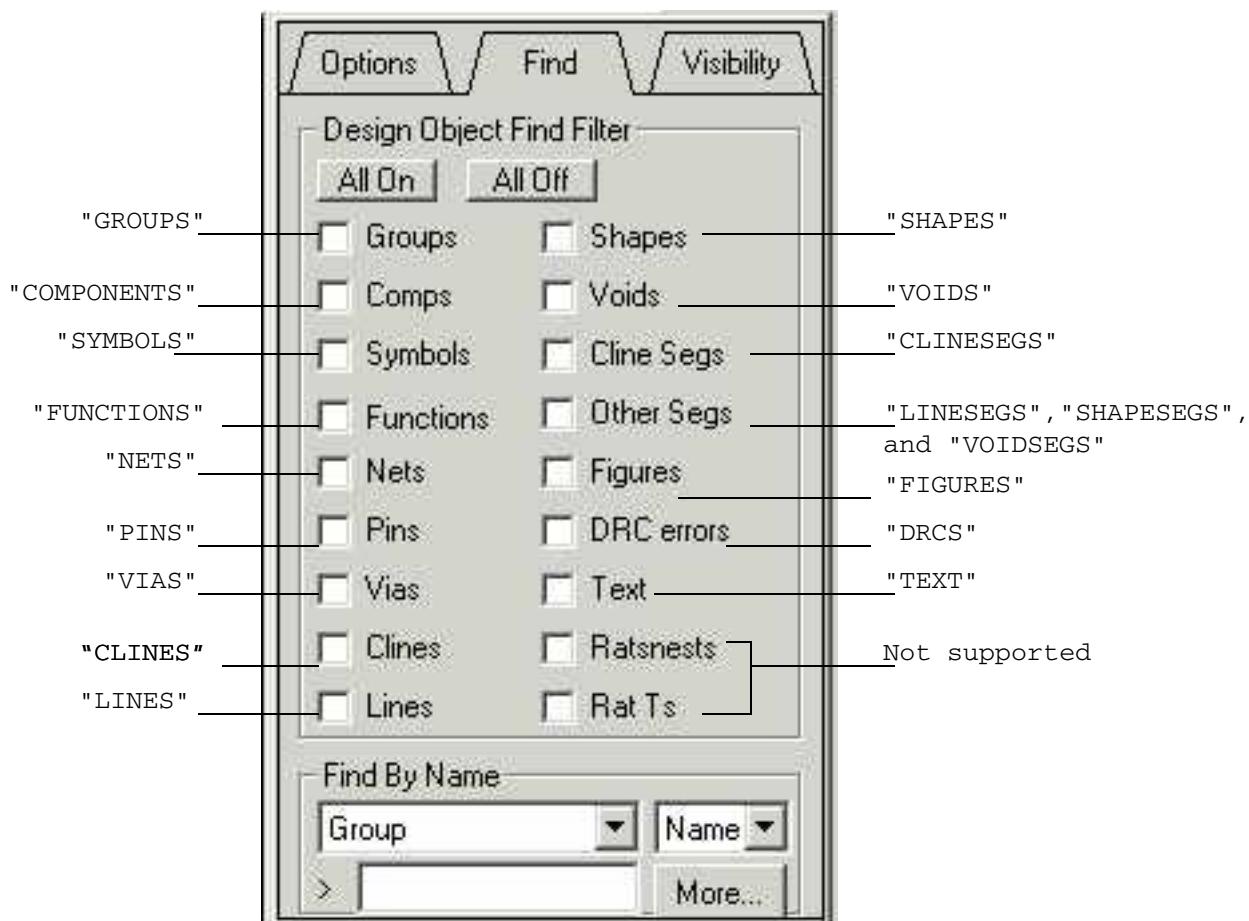
## Allegro SKILL Reference

### Selection and Find Functions

example, to move the shape). If you want to access information on the shape, then do not set the BOUNDARY\_SHAPES option (for example, to do show element on the shape).

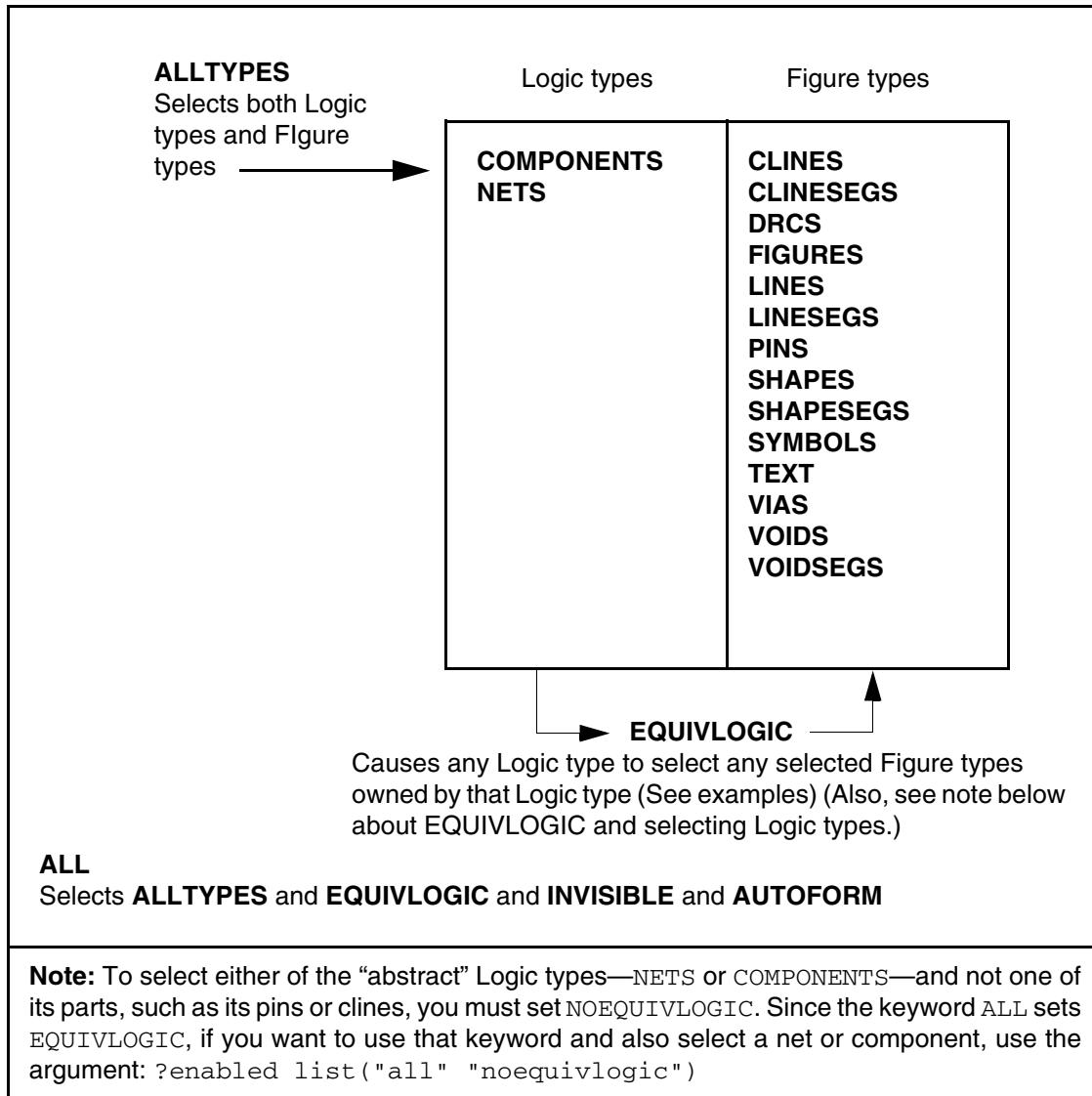
**Note:** If you pass "ALL" or "ALLTYPES" to *setOptions*, then BOUNDARY\_SHAPES will be enabled and user's selecting a ETCH layer generated shape will result in the selection returning its dynamic shape on the "BOUNDARY CLASS". If you wish to select the generated shape, but want to use the *all* option, use "(ALLTYPES" "NOBOUNDARY\_SHAPES" . You need only pass BOUNDARY\_SHAPES to the enabled list. It will be ignored if passed to the onButtons list.

**Figure 4-1 Find Filter and Related Keywords**



The differences in effects of different combinations of keywords can be subtle. [Figure 4-2](#) on page 317 shows the relationship between the keywords.

**Figure 4-2 Relationship Among axlSetFindFilter Keywords (See examples also)**



### Bond Objects

Bond objects (bond wires and fingers) are enabled only in APD/SIP. These options are ignored in the PCB products.

In APD/SIP, to maintain backward compatibility, an application is not considered to be "bond smart". This means if the SKILL application enables VIAs then FINGERS are enabled and if CLINES is enabled, the BOND WIRES are automatically enabled.

To make an application bond smart, you either set the BONDSMART option or use the BONDWIRES, NOBONDWIRES, FINGERS or NOFINGERS options. An application that is bond smart can separately control the selection of bond objects (fingers or bond wires) from their base objects (vias or clines).

**Note:** You only need to make your SKILL application bond smart if you wish to differentiate the disabling of one of these objects. By default, users in APD/SIP will be able to independently select: VIAS or FINGERS if the SKILL application enables VIAS or CLINES and WIRES if CLINES are enabled. Ideally most applications will not need to be updated to be bond smart.

## 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 <b>Find Filter</b> 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.
<code>lt_onButtons</code>	List of keyword strings that describe object types that are to be enabled for selection. Enabled types will appear with the <code>onButton</code> depressed in the Find Filter form. <code>onButton</code> 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 <code>on</code> in both the enabled and the <code>onButton</code> lists to be fully enabled for selection. Options are ignored when provided in the <code>onButton</code> 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

<code>t</code>	One or more Find Filter changes were made.
<code>nil</code>	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 same command was used. 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`.

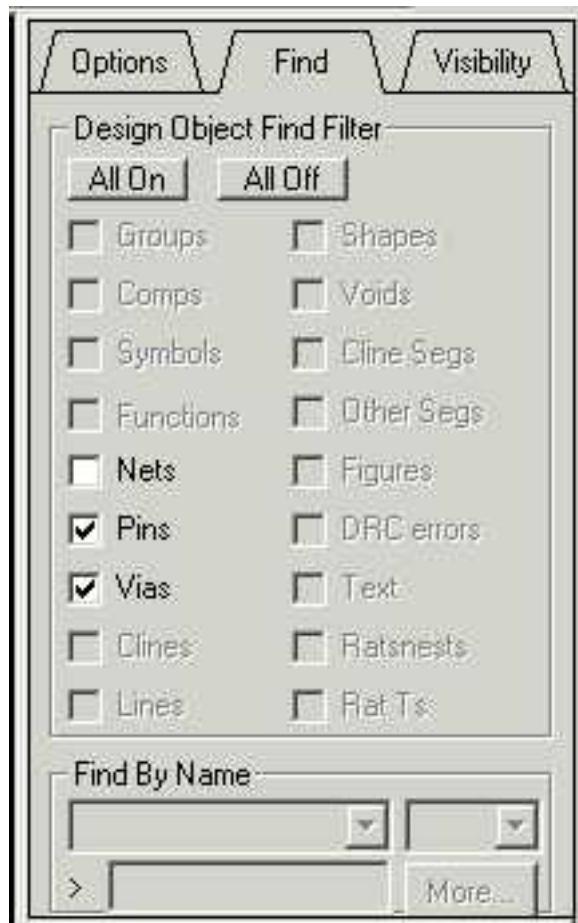
## Allegro SKILL Reference

### Selection and Find Functions

#### Example 1

```
(axlSetFindFilter ?enabled (list "vias" "pins" "nets")
?onButtons (list "vias" "pins"))
```

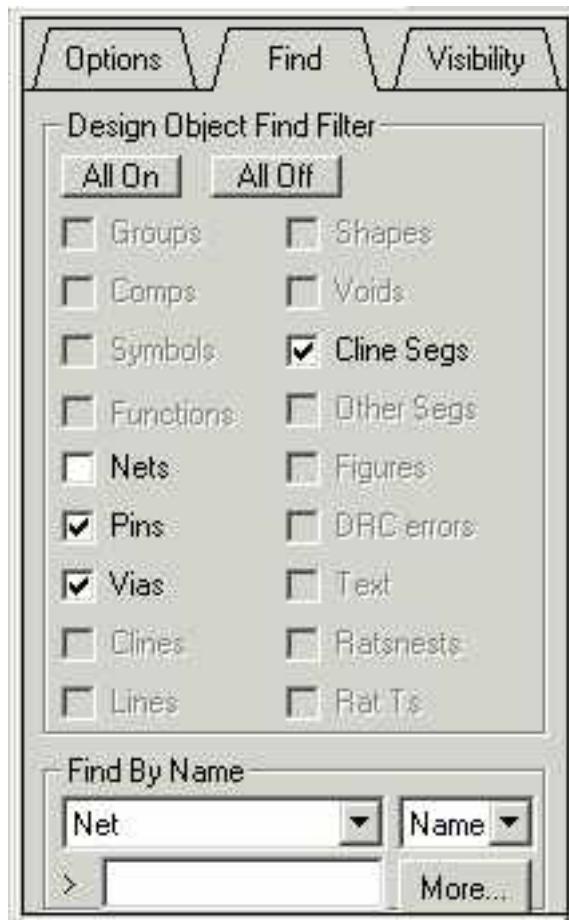
Displays the Find Filter with a list of Nets, Pins, and Vias. The Pins and Vias boxes are turned on as shown:



## Example 2

```
(axlSetFindFilter ?enabled (list "noall" "vias" "pins"  
    "nets" "clinesegs" "nameform")  
?onButtons (list "vias" "pins" "clinesegs"))
```

Displays the Find Filter with Pins, Vias, and Cline segs 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.

## axlDBFindByName

```
axlDBFindByName (  
    s_type  
    t_name  
) ⇒ o_dbid/nil
```

### Description

Finds `dbid` of an object by name without involving the selection set. This means you can run this command any time without affecting what exists in the selection set. The following object lookup types are supported:

<b>type</b>	<b>t_name</b>	<b>return type</b>
'net	net name	<code>dbid</code> of a net
'refdes	refdes	<code>dbid</code> of a component instance
'padstack	padstack name	<code>dbid</code> of a padstack

This is restricted to a single object. To find objects using wildcards, use `axlSelectByName`.

### Arguments

- s\_type*      Type symbol; see above.  
*t\_name*      Object's name (this is case insensitive).

### Value Returned

- `o_dbid`      `dbid` of object or nil.

### See Also

[axlSelectByName](#)

### Examples

```
db = axlDBFindByName('net "GND")  
db = axlDBFindByName('padstack "VIA")
```

## **Allegro SKILL Reference**

### Selection and Find Functions

---

```
db = axlDBFindByName('refdes "U1")
```

## **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 / !t_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:

<b>Object Type</b>	<b>Item the function finds</b>
"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
"NETCLASS"	find netclass by name
"NET_GROUP"	find a net group by name
"REGION"	find region by name

## Allegro SKILL Reference

### Selection and Find Functions

---

Object Type	Item the function finds
"XNET"	find xnet by name; Will return a Net if xnet is a single net xnet. See <cdsroot>/share pcb/examples/skill/select/ashfindxnet.il
"MATCH_GROUP"	find match group by name
"MODULE"	find module by name

You can use wildcards with this function:

- " \* " matches any sequence of zero or more characters.
- " ? " matches any single character.
- " \ " is used to send a special character, for example, \x.

Provided third argument [*g\_wildcard*] is set to TRUE.

**Note:** This saves and restores the current find filter settings, but resets the selection set.

## Arguments

<i>t_objectType</i>	Type of database name.
<i>t_name</i>	Object to find.
<i>lt_name</i>	List of names to find.
<i>[g_wildcard]</i>	If * or ? appear in name, use regular expression matching

## Value Returned

<i>t</i>	List of objects found.
<i>nil</i>	No matching name or illegal <i>objectType</i> 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)
```

## **Allegro SKILL Reference**

### Selection and Find Functions

---

```
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.*" t)  
(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.?" t)  
(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.

#### **Example 13**

```
Skill > p = axlSelectByName ("NET" "N*" t)  
(dbid:28630044 dbid:28634000 dbid:28638750)
```

Finds nets starting with N.

## axlSelectByProperty

```
axlSelectByProperty(
    t_objectType
    t_property
    [t_value]
    [g_regularExpression]
)
⇒ lo_dbid/nil
```

### Description

Selects the *dbid* set of a particular Allegro 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 Allegro PCB Editor differ from the SKILL regular expressions. Allegro PCB Editor handles regular expressions such that they are more compatible with the character set allowed in Allegro 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.

Properties must be upper case. The value is case insensitive if wildmatch match is `nil` while case sensitive for wildcard matching.

**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 Allegro PCB Editor database object type. Must be: compdef, component, drc, net, symdef, symbol, or group.
<i>t_property</i>	Name of property.
<i>t_value</i>	Optional property value.
<i>g_regularExpression</i> <i>on</i>	<i>t</i> if property value is to be treated as a regular expression, or <i>nil</i> , which is the default, to treat property value as a simple match.

## Value Returned

<i>t</i>	One or more <i>dbids</i> added to the select set.
<i>nil</i>	No <i>dbids</i> 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.

## **axISnapDisableAtRMB**

```
axISnapDisableAtRMB (  
    )  
    ⇒ t
```

### **Description**

Disable the display of "snap to" submenu at right-click popup.

Use in conjunction with [axISnapEnableAtRMB](#) to remove the "Snap To" from the right-click popup.

### **Arguments**

None

### **See Also**

[axISnapEnableAtRMB](#)

### **Value Returned**

t	Always returns t
---	------------------

## **axISnapEnableAtRMB**

```
axISnapEnableAtRMB()  
)  
⇒ t
```

### **Description**

Enable the display of "snap to" submenu at right mouse button popup.

It is used to enable the display of "snap to" submenu at right-click popup. The function should be used with [axISnapDisableAtRMB](#) to achieve "snap to" submenu which is available with all the popups of an interactive command. To enable or disable "snap to" submenu at particular steps during execution, this function can be used with [axISnapDisableAtRMB](#).

This is the default mode with all axIEvent APIs with a right mouse button.

### **Arguments**

None.

### **Value Returned**

t	Always returns t
---	------------------

### **See Also**

[axISnapDisableAtRMB](#), [axIEnterEvent](#)

### **Example**

See <cdsroot>/share pcb/examples/axlcore/enterevent.il

## axlSnapToObject

```
axlSnapToObject(  
    g_mode  
    xy  
)  
⇒ xy/nil
```

### Description

Supports snapping to a logic object's connect point. A logic object is a:

- Cline
- Clines segments (lines or arcs)
- Via
- Pin

For cline objects, the two end points are considered the connect points. For pad objects, the connect point is defined in the padstack. Snapping is based upon the trap size, which varies based upon the zoom factor (see `axlGetTrapBox`). The smaller the trap size, the higher the zoom. If no object is found within the original xy location is returned.

**Note:** Avoid using the grid snapping option with `axlEnterPoint` as it might move the user pick outside the trap size.

### Arguments

*g\_mode*      *nil*: Use active sel set to determine snapping.  
*t*: Snap based upon the visible layers in the database.  
*xy*              Location in design units to snap (list of design units x:y).

### Value Returned

*xy*              Object snapped point (list of design units x:y).  
*nil*              If not object exists for snapping returns nil.

Also nil if arguments are incorrect

## See Also

[axlGetSelSet](#), [axlGetTrapBox](#), [axlGetLastEnterPoint](#)

## Examples

Pseudo code to move objects.

```
; select object(s); do not do a axlClearSelSet
    _clpSelect
        ; first snap to an object that is the selection list
        gridSnap = axlEnterPoint(?prompts list("Pick origin point")
                                ? gridSnap t)
; for better results use the unsnapped pick
    origin = axlSnapToObject(nil axlLastPick(nil))
    ; if no object found fallback to the grid snapped pick
    unless( origin origin = gridSnap)
; ok to do clear now
    axlClearSelSet()
; now ask user for a destination
    gridSnap = axlEnterPoint(?prompts list("Pick origin point")
                            ? gridSnap t)
; Now snap the destination point based upon what can be found at that
    ; location
    ; for better results use the unsnapped pick
    dest = axlSnapToObject(nil axlLastPick(nil))
    ; fallback to grid snapped location if nothing found
    unless( origin dest = gridSnap)
; modify coordinates with dest location
    axlTransformObject(....)
```

## **axlLastPickIsSnapped**

```
axlLastPickIsSnapped()  
-> t/nil
```

### **Description**

Normally called after an `axlEnter` call to determine if the pick was snapped or unsnapped.

### **Arguments**

none

### **Value Returned**

`t` if last picked was snapped, `nil` if unsnapped

### **Examples**

```
snappedPoint = axlEnterPoint(?prompts list("Pick origin point")  
    ?gridSnap t)  
state = axlLastPickIsSnapped()
```

## **Allegro 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 Allegro 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 Allegro PCB Editor are an example of `readOnly` Allegro PCB Editor figure objects. An AXL program cannot modify DRC objects directly.

## AXL/SKILL Interactive Edit Functions

This section lists interactive edit functions.

### **axlAllShapesMerge**

```
axlAllShapesMerge(  
    l_mode  
    l_layer  
)  
==> nil
```

#### **Description**

Merge all shapes, modify all dynamic shapes to static before merge.

#### **Arguments**

- |                |  |
|----------------|--|
| <i>l_mode</i>  | <ul style="list-style-type: none"><li>■ 0: merge all shapes</li><li>■ 1: merge all dynamic shapes which will be converted to static shapes before merge</li><li>■ 2: merge all static shapes</li></ul> |
| <i>l_layer</i> | <ul style="list-style-type: none"><li>■ -1: merge all shapes on all layers</li><li>■ n: merge all shapes on layer "n" (n &gt;= 0)</li></ul>  |

#### **Value Returned**

*l\_result/l\_layer*: Returns nil

#### **Example**

- merge all shapes  
`axlAllShapesMerge(0 nil)`
- merge all dynamic shapes  
`axlAllShapesMerge(1 nil)`

## **axlBondFingerDelete**

```
axlBondFingerDelete (
    bondFingers
    deleteWires
)
==> t/nil
```

### **Description**

Deletes the (list of) bond fingers passed in. Optionally, it will delete the connect bond wire elements as well.

### **Arguments**

bondFingers	either a dbid or a list of dbids representing the bond fingers to be deleted.
deleteWires	t/nil to tell the system whether it should remove any bond wires connected to the fingers.

### **Value Returned**

Value is *t* returned, if one or more objects are deleted; otherwise the return value is *nil*.

## **axlBondWireDelete**

```
axlBondWireDelete(  
    bondWires  
    deleteFingers  
)  
==> t/nil
```

### **Description**

Deletes the (list of) bond wires passed in. Optionally, it will delete the connect bond finger elements as well.

### **Arguments**

<b>Argument...</b>	<b>Value...</b>
bondWires	dbid or list of dbid, representing the bond wires to be deleted.
deleteFingers	<ul style="list-style-type: none"><li>■ t: Bond fingers connected to the wires are removed</li><li>■ nil: Connect bond fingers are not deleted</li></ul>

### **Value Returned**

Value is t returned, if one or more objects are deleted; otherwise the return value is nil.

## **axlChangeLine2Cline**

```
axlChangeLine2Cline(  
    lo_dbid/o_dbid  
)  
==> x_cnt/nil
```

### **Description**

Changes provided lines to clines. Lines not on an etch layer are ignored. If a line is converted to a cline then it may be assigned to a net, otherwise it will be left on a the standalone branch.

### **Arguments**

*lo\_dbid/o\_dbid*      A single dbid or list of line dbids

### **Value Returned**

*t* if succeeded, *nil* if failure

FAILURES: (for debug purposes set axlDebug(*t*) to see additional messages)

- dbid is not a line or a line on ETCH class
- line is LOCKED or FIXED

### **Examples**

#### ■ Convert a line

```
res = axlDBCreateLine('(0:0 100:100) 5 "ETCH/TOP")  
res = car(res)  
cnt = axlChangeLine2Cline(res)
```

### **See Also**

[axlTransformObject](#)

## **axlChangeLineFont**

```
axlChangeLineFont(  
    o_dbid  
    x_newFont  
)  
==> lo_dbid/nil
```

### **Description**

Changes font on a line or segment.

### **Arguments**

<i>o_dbid</i>	A line dbid
<i>x_newFont</i>	The new font. The valid values are 'SOLID' 'HIDDEN' 'PHANTOM' 'DOTTED' 'CENTER'
	nil is same as SOLID

### **Value Returned**

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

### **Examples**

#### ■ Changes the font of a line to hidden

```
; ashOne is a selection utility found at <cdsroot>/share pcb/examples/skill/  
ash-fxf/ashone.il  
dbid = ashOne()  
; pick a line, cline or segment (set find filter)  
updatedDbid = axlChangeLineFont(dbid 'hidden')
```

### **See Also**

[axlTransformObject](#), [axlChangeLayer](#)

## axlChangeWidth

```
axlChangeWidth(  
    lo_dbid/o_dbid  
    f_newWidth  
    [g_invisible]  
)  
==> lo_dbid/nil
```

### Description

Changes width of lines, clines and segments (arc and line).

By default, only visible lines are changed. This allows layer filtering by temporary changing the visible layers (see example in [axlVisibleUpdate](#)). If you wish to override this behavior then set the value of the optional variable *g\_invisible* to t.

**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

<i>lo_dbid/o_dbid</i>	Single dbid or list of dbids.
<i>f_newWidth</i>	New width of line.
<i>g_invisible</i>	If t objects do not need to be visible on the display to have their width changed.

### 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](#), [axlChangeLayer](#)

## **axlCopyProperties**

```
axlCopyProperties(  
    o_destDbid  
    o_srcDbid  
) ==> t/nil
```

### **Description**

This copies properties from one object to another. It filters certain properties. If you need to copy properties suggest you utilize this interface instead doing it yourself.

Side effects may happen when properties are copied.

Properties filtered are:

- IDX family
- IDF\_OWNER
- CLIP\_DRAWING
- FIXED
- FIXED\_PRIVATE
- DYN\_SHAPE\_PRIORITY
- SUBNET\_NAME only if shape is not on ETCH

Also properties may not be copied if they do not meet the rules for the destination element (example not legal for element type).

Existing properties are maintained on destination object but they may be overridden by source object.

### **Arguments**

*o\_destDbid*      Destination for properties  
*o\_srcDbid*      Source for properties

### **Value Returned**

*t* if objects are dbids, *nil* if one or more object is not a dbid.

## **Allegro SKILL Reference**

### Interactive Edit Functions

---

#### **See Also**

[axlCopyObject](#)

## axlCopyObject

```
axlCopyObject(
    lo_dbid/o_dbid
    ?move          l_deltaPoint
    ?mirror        t/nil
    ?angle         f_angle
    ?origin        l_rotatePoint
    ?allOrNone     t/nil
    ?retainNet     t/nil
)
==> t/nil
```

### Description

Use this function to copy the database object(s). This supports the same functionality as [axlTransformObject](#) except it copies and transforms one or more objects.

One additional option supported is retainNet. This only applies to vias. If the value of this option is set to `t`, the net of the via is retained on copy, `nil` allows the via to connect to whatever it touches at the new location. In the board, pins are not supported.



#### Caution

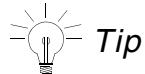
Properties and text attached to the database object are also copied. Also [see axlTransformObject cautions](#).

### Arguments

lo_dbid/o_dbid	a single dbid or list of dbids
l_deltaPoint	optional move distance
mirror	optional mirror object (see above table)
f_angle	optional rotation angle
l_rotatePoint	optional rotation point
allOrNone	if <code>t</code> and a group of objects, transform must succeed on all objects or fail
retainNet	<code>t/nil</code> (applies to vias only)

## Value Returned

list of transformed objects or `nil` if failed.



If you need to copy a group of objects the performance is much better if you call this function with the object group instead of passing each dbid individually.

## Examples

`lDbid` = list of database objects

`dbid` = one database object

**Case 1:** Copy a set of objects 1000 database units vertically

```
r = axlCopyObject(lDbid, ?move '1000.0:0.0)
```

**Case 2:** Copy and rotate an object about its origin 45 degrees

```
r = axlCopyObject(dbid, ?angle 45)
```

**Case 3:** Copy and rotate an object about a rotate point

```
r = axlCopyObject(dbid, ?angle 45 ?origin 100:100)
```

## See Also

[axlTransformObject](#), [axlDBCloak](#), [axlCopyProperties](#)

## axlDBAltOrigin

```
axlDBAltOrigin(  
    g_mode  
    o_dbid  
)⇒ xy/nil
```

### Description

Returns alternative center for a *dbid*. This provides SKILL access to the `move` command's origin point option in the Options tab.

It is intended for symbols instances (it will convert a component instance to its symbol). Body origin rules for symbols, origin is the first rule that is met:

1. the origin of text on the PACKAGE\_GEOMETRY/BODY\_CENTER layer
2. the center of an extent box created by the union of all shapes on layers  
PACKAGE\_GEOMETRY (PLACE\_BOUND\_TOP, PLACE\_BOUND\_BOTTOM,  
DFA\_BOUND\_TOP, DFA\_BOUND\_BOTTOM) and EMBEDDED\_GEOMETRY  
(PLACE\_BOUND and DFA\_BOUND)
3. center of the symbol bbox

Other Allegro figure *dbids* can be supplied, but all options may not be supported. For example, a CLINE supports the center option, but not '`origin`' or '`pin1`'.

## Arguments

- g\_mode*      The '*center*' option returns the body center of an object.  
                The '*origin*' option returns the origin of an object (normally if *dbid* has an *xy* attribute, this is the same coordinate).  
                For Symbols, the board origin can be set by the origin of text on the PACKAGE\_GEOMETRY/BODY\_CENTER layer.  
                The '*pin1*' option returns pin1 as center.
- o\_dbid*      A figure (geometric object) *dbid*.

## Value Returned

- xy*      Location requested.  
*nil*      Not a *dbid*, *dbid* is not a figure dbid, or mode is not supported for that object.

## See Also

[ax1DBGetSymbolBodyExtent](#)

## Example

The ;ashOne utility is supplied in the examples SKILL code.

```
sym = ashOne()  
;  
; select symbol in Find filter and select a symbol  
sym->xy  
;  
; prints (3503.0 1058.0)  
ax1DBAltOrigin('origin sym)  
;  
; prints (3503.0 1058.0) -- origin of symbol is same as xy  
ax1DBAltOrigin('center sym)  
;  
; prints (3250.0 1737.0)  
ax1DBAltOrigin('pin1 sym)  
;  
; prints (3503.0 1058.0) -- pin1 of symbol is same as its x
```

## **axlDBChangeText**

```
axlDBChangeText(  
    o_dbid  
    t_text  
    [r_textOrientation/x_textBlock]  
)  
==> l_result/nil
```

### **Description**

Modifies the characteristics of a text string in the layout. To keep current settings on the text, set the arguments, `t_text` and `r_textOrientation` to `nil`. To move text use the [axlTransformObject](#) object.

**Note:** For renaming refdes this works the same as edit text in that it checks for the `HARD_LOCATION` property and will not rename refdes if this property is present. If you want to ignore this property, use [axlRenameRefdes](#).

### **Arguments**

<code>o_dbid</code>	Database ID of text
<code>t_text</code>	Text string.  If the value of this argument is set to <code>nil</code> , current settings are retained on the text
<code>r_textOrientation</code>	Orientation of text  The <code>nil</code> value indicates that current settings are to be retained on the text.  <a href="#"><u>See Structure.</u></a>
<code>x_textBlock</code>	To be specified if only the text block is to be changed

### **Structure**

The axlTextOrientation structure is as follows.

```
defstruct axlTextOrientation
  r_textOrientation ; orientation of text
    textBlock;      A string specifying the text block name
    rotation;      A floatnum variable specifying rotation in degrees
    mirrored;      Possible values are:
                  □ t: mirrored
                  □ nil: not mirrored
                  □ GEOMETRY: only geometry is mirrored.
    justify;       Supported values:
                  □ left
                  □ center
                  □ right
```

If any of these arguments is modified, then you need to provide values for all arguments. Arguments for which the values are not changed, copy the values from the existing text dbid.

**Note:** As with all SKILL defstructs, use constructor function, make\_axlTextOrientation to create instances of axlTextOrientation. To copy instances of axlTextOrientation, use the copy function, copy\_axlTextOrientation.

### **Value Returned**

nil	if not created
I_result	List containing: <ul style="list-style-type: none"><li>■ (car) list of DBID of the text</li><li>■ (cadr) t if DRCs created or nil.</li></ul>



***Do not pass text string with newlines as an argument.***

## See Also

[axlTransformObject](#), [axlChangeLayer](#), [axlRenameRefdes](#), [axlTextOrientationCopy](#)

## Example

Example is text added in axlDBCreateText

```
text = car(ret)
```

- Change text

```
cret = axlDBChangeText(text "Chamfer neither sides")
```

- Change text block

```
cret = axlDBChangeText(text nil 4)
```

- Change rotation and text

```
axlTextOrientationCopy(text myorient)  
myorient->rotation = 0.0  
cret = axlDBChangeText(text "New text" myorient)
```

## **axlDBConnectItem**

```
axlDBconnectItem(  
    o_dbid  
    [x-trapSize]  
)  
⇒ x_count/nil
```

### **Description**

Connects an unconnected cline end (path) to close by pins or vias. It basically does a device connectivity to connect clines to vias.

By default, if trap is absent or nil use the cline width to construct a search box of  $2 * \text{width}$ . This is normally sufficient if the cline ends on the pad or within the pad to make a connection. If there are multiple pads within the search box it will use the closest pad to the unconnect end based on the origin of the pad.

To achieve a connection it may:

- extends a segment
- adds a new segment
- shortens a segment (occurs if the segment crosses the pad's connect point and does not make a connection). This is atypical.

### **Notes**

- Currently does not consider net. Future versions may take net into account.
- If changing many clines put the operation in `axlDBCloak`.
- It is recommend to use the default `trapSize`. Large values may slow performance.

### **Arguments**

<code>o_dbid</code>	A dbid, path (cline)
<code>x_trapSize</code>	Optional size to look in design units. By default, uses the cline width to construct a box of $\text{width} * 2$ .  Trapsize is limited to 4 times the line width.

## Value Returned

nil	An illegal argument
x_count	Number of connections made.
	Can be 0 (no additional connections) or 1 or 2 new connections

## See Also

[axlDBGetConnect](#)

## Examples

Uses "ash" example software in example hierarchy

```
cline = ashOne('("NOALL" "CLINES"))
ret = axlDBConnectItem(cline)
```

## **axlDeleteObject**

```
axlDeleteObject(  
    o_dbid/lo_dbid  
    [g_mode]  
)  
⇒ 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 leaves the physical objects.

Command allows for rip-up of associated etch via the ripup option.

```
axlDeleteObject(lo_dbid 'ripup)
```

Except for Nets, objects will be erased before they are deleted. Only the Net's Ratsnests is erased. Other parts of a Net will not be erased because there is no ripup. If a Net is in a highlighted state, it will be dehighlighted.

Also allows deletion of the following parameter records:

- artwork (films)

Both individual films can be deleted and all films. If all films are deleted then next time the artwork dialog is opened then it will be auto-populated with the default films.

- subclasses

subclasses must be empty and legal for deletion (cannot delete PIN subclasses).

In the case of deleting parameter records, the current restriction is to only pass that single object. Do not try to pass multiple parameter objects or to mix them with non-parameter objects.

## Arguments

<i>o_dbid/lo_dbid</i>	<i>dbid</i> , or list of <i>dbids</i> to delete from layout.
<i>g_mode</i>	optional delete options.'riput - enable etch ripup option (same as Allegro delete ripup command ripup option)

## Value Returned

<i>t</i>	Deleted one or more objects from the layout.
<i>nil</i>	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 symbols for components). To emulate the Allegro delete command behavior, select and 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())))
  )
```

## **Allegro SKILL Reference**

### Interactive Edit Functions

---

```
axlUIPopupSet( axlUIPopupDefine(nil nil))  
))
```

The following deletes the TOP artwork film record

```
p = axlGetParam("artwork:TOP")  
axlDeleteObject(p)
```

The following deletes all films

```
axlDeleteObject(axlGetParam("artwork"))
```

## **axlDeleteByRectangle**

```
axlDeleteByRectangle
  g_windowSelect
  g_windowDelete
  g_insideOutside
  [b_ignoreFixed]
  [l_objectTypes]
-> t/nil
```

### **Description**

This function removes elements from a design relative to a rectangular box provided (*g\_windowDelete*). The box is defined by two points on opposite sides of the box and the edges of the box will be parallel to the x and y axes. *g\_windowSelect* is another box which is used to select items for removing. It is a SKILL access to the *Manufacture – Drafting – Delete by Rectangle* command in the tool.

The interface operates by removing all objects either inside or outside of the provided rectangle. For those elements which cross the edge of the window provided, the object is cut there. This means only part of a shape over top of the rectangle will be removed, and a cline will be cut at the XY location where it meets the rectangle (the center line will be cut at the box boundary, but the end caps of the trimmed cline may still extend over the boundary).

## Arguments

<i>l_windowSelect</i>	The parameter type is a list of 2 points defining a box to select items which need to be cut, for example, list(-500:-500 500:500)
<i>l_windowDelete</i>	Bounding box to be cut relative to. For example, list(-500:-500 500:500)
<i>s_insideOutside</i>	'inside: Deletes elements inside the box, while keeping those outside of it.  'outside: Deletes elements outside the box, while keeping those inside of it.
<i>[b_ignoreFixed]</i>	t: Ignores fixed property on elements to allow them to be deleted.  nil: Respects the fixed property, so fixed elements will not be deleted, even though they are in an area where non-fixed elements are deleted. If not provided, default value is nil.
<i>[ls_objectTypes]</i>	The argument type is a list of strings, or a single string if there is only one object type to be deleted. List may include any set of: "VIAS" "FINGERS" "CLINES" "LINES" "BONDWIRES" "SHAPES" "CLINESEGS" "OTHERSEGS". If not provided, all object types will be processed.

## Value Returned

<i>t</i>	Operation was successful.
<i>nil</i>	Operation failed.

## Example

```
axlDeleteByRectangle(list(-500:-500 500:500) list(-500:-500 500:500) 'inside t  
' ("VIAS" "CLINES"))
```

## See Also

[axlDeleteObject](#)

## **axlDeleteTaper**

```
axlDeleteTaper(  
    o_dbid  
)  
==> t/nil
```

### **Description**

Deletes tapers

### **Arguments**

*o\_dbid*      dbid of Shape or PATH.

### **Value Returned**

- *t* – indicates success
- *nil* – command failed

## 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 <i>nil</i> , 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 <i>nil</i> .
<i>nil</i>	No properties deleted.

### See Also

[axlDBAddProp](#), [axlDBDeletePropAll](#)

## Allegro SKILL Reference

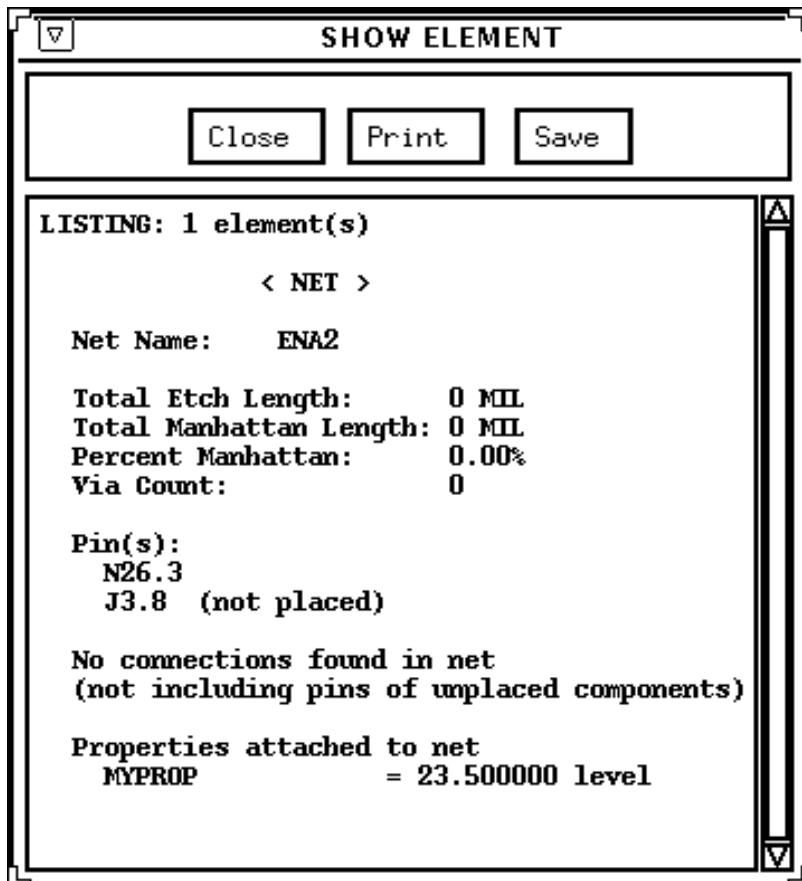
### Interactive Edit Functions

#### 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.



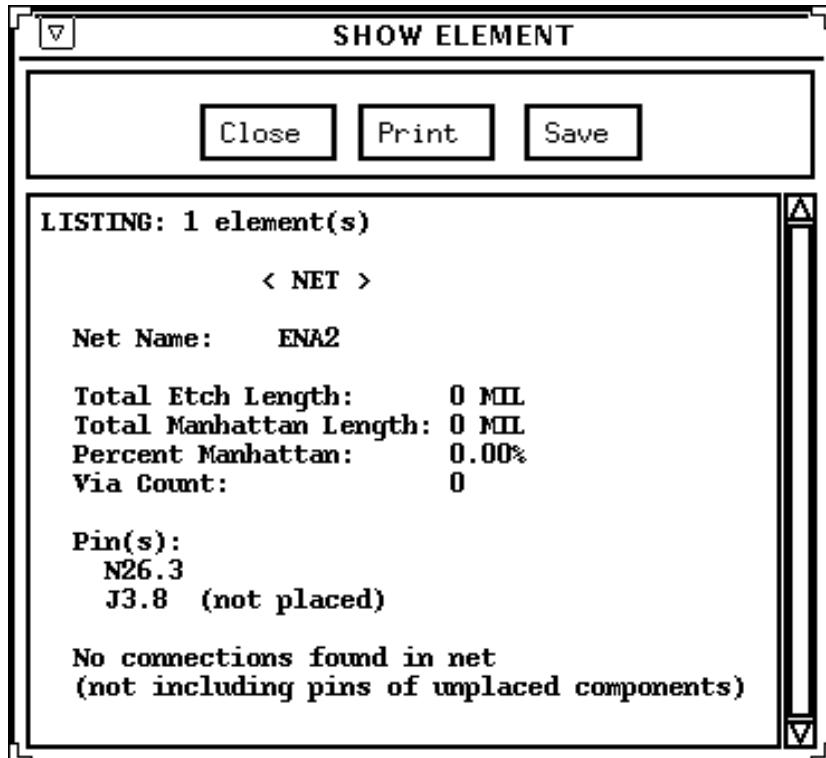
## Allegro SKILL Reference

### 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.



## **axlDBDeletePropAll**

```
axlDBDeletePropAll(  
    t_name  
)  
==> x_count/nil
```

### **Description**

Deletes all instances of the property *t\_name* in the database. This includes properties that exist on the symDef and compDef that cannot be access via the property edit command. If you delete a property that effects the DRC system, you may wish to wrap this call with a axlDBCloak for better performance.

### **Arguments**

*t\_name*      Name of property to have all its instances deleted.

### **Value Returned**

<i>x_count</i>	Returns number of properties deleted
<i>nil</i>	Error, property definition doesn't exist

### **See Also**

[axlDBAddProp](#), [axlDBDeleteProp](#), and [axlDBCloak](#)

### **EXAMPLE**

Delete all fixed properties in database

```
axlDBDeletePropAll ("FIXED")
```

## **axlDBDeletePropDictEntry**

```
axlDBDeletePropDictEntry(  
    t_name  
)  
==> t/nil
```

### **Description**

Deletes an unused user property definition. Property entry must be unused. The property definition must be a user property and its useCount (`axlDBGetPropDictEntry`) must be zero for you to delete it. Use `axlDBDeletePropAll` if property is in use.

### **Arguments**

*t\_name*      String specifying the name of the user property dictionary entry to be deleted.

### **Value Returned**

*t* :      Deleted the property definition.

*nil*      Property is in use, is an Allegro property, property does not exist, or name is not legal.

### **See Also**

[axlDBAddProp](#), [axlDBDeleteProp](#), and [axlDBCreatePropDictEntry](#)

### **EXAMPLE**

take property, myprop, created axlDBCreatePropDictEntry

```
    axlDBDeletePropDictEntry("myprop")
```

## **axIDBOpenShape**

```
axIDBOpenShape(  
    o_shapeDbid/nil  
    [o_polygon/r_path/nil]  
    [g_close]  
)  
==> o_dbid/nil
```

### **Description**

Opens an existing shape to replace its boundary or to modify its voids.

Shape can be left open so you can update the voids within the shape. If only the outline needs to be replaced, you can close the shape as part of this call. The new outline cannot overlap existing voids or allow existing voids to exist outside the outline.

**Note:** A side-effect of opening an existing shape is the shape will be displayed as unfilled until it is closed.

### **Arguments**

o_shapeDbid	dbid of shape to be modified. If dbid is nil then use the existing open shape
o_polygon	new shape outline in polygon format
r_path	new shape outline in r_path format
g_close	optional option to close the shape (t) boundary modification

### **Value Returned**

o\_dbid      dbid of provided shape or nil if an error

### **See Also**

[axIDBCreateCloseShape](#), [axIDBCreateOpenShape](#), [axIDBCreateVoid](#),  
[axIShapeDeleteVoids](#), [axIShapeAutoVoid](#)

## Examples

ashOne is a shareware utility that allows user to select an object (see *<CDSROOT>/share/pcb/examples/skill/ash-fxf/ashone.il*)

### 1. Select a shape and expand it by 100

```
shp = ashOne("shapes")
edge = car( axlPolyFromDB(shp) )
newedge = car( axlPolyExpand(edge 100.0 'NONE) )
newshp = axlDBOpenShape(shp newedge t)
```

### 2. Select a void delete it

```
shp = ashOne("voids")
edge = axlPolyFromDB(shp)
newedge = car( axlPolyExpand(edge 100.0 'NONE) )
newshp = axlDBOpenShape(shp newedge)
q = axlDBCreateCloseShape(newshp)
```

### 3. Select a shape, delete all voids and contract boundary by 100

```
shp = ashOne("shapes")
edge = car( axlPolyFromDB(shp) )
newedge = car( axlPolyExpand(edge -100.0 'NONE) )
newshp = axlDBOpenShape(shp nil)
axlShapeDeleteVoids(shp)
q = axlDBCreateCloseShape(newshp)
```

## **axlDeleteFillet**

```
axlDeleteFillet(  
    o_dbid  
)  
⇒ t/nil
```

### **Description**

Deletes fillet associated with a PIN, VIA, T, or CLINE. The command also deletes a single fillet if *o\_dbid* is a fillet shape.

When deleting via a cline, Allegro PCB Editor searches for the via/pin connections and deletes the fillets from that pin or via. It only deletes FILLETS on the layer of the CLINE. If deleting FILLETS from a PIN or VIA, it deletes FILLETS on all layers.

### **Arguments**

*o\_dbid*      *dbid* of a PIN, VIA, PATH, or T.

### **Value Returned**

t      Fillet deleted.

nil      No fillet deleted.

## axlFillet

```
axlFillet (
    o_dbid/lo_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, Path(CLINE) or  
                  line(segment)

### Value Returned

t	Fillet(s) added.
nil	Error or 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" )) )
```

## **axlFilletConvert**

```
axlFilletConvert(  
    o_dbid  
) -> t/nil
```

### **Description**

Converts a fillet or taper to a static shape.

This command should only be used if converting a design to another form such as if you are crafting a panelization solution.



Dynamic fillets should be disabled when using this command.

Some of the side effects of using this command are:

- The converted fillets remain if etch is deleted/modified
- If dynamic fillets are enabled, a duplicate fillet appears.
- DRCs may occur because the converted fillet or taper uses shape spacing.
- Voiding other shapes may change.
- Etch length calculations may be effected.
- Etch editing may work differently on these traces.

### **Arguments**

*o\_dbid*      A fillet shape dbid.

### **Value Returned**

*t* if the fillet is successfully converted to a static shape, *nil* if dbid is not a fillet shape.

### **See Also**

[axlDeleteFillet](#)

## **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).

## **axlLastPick**

```
axlLastPick(  
    l_mode  
)⇒ xy
```

### **Description**

This returns the last processed cursor pick. You can snap to current grid (*l\_mode* ⇒ t) or leave it unsnapped. Position is returned in design units. The grid used depends on the active layer. A pick event causes the last pick. In SKILL, a call to axlEnterPoint, axlEnterEvent, and so on may generate this. It allows switching from a snapped to an unsnapped event. If a user has made no pick since launching Allegro PCB Editor, then it returns (0 0).

### **Arguments**

*l\_mode*      t for snapped and nil for unsnapped.

### **Value Returned**

Last pick as an xy list.

### **Examples**

```
snappedPoint = axlEnterPoint(?prompts list("Pick origin point") ?gridSnap t)  
unsnapped = axlLastPick(nil)
```

## **axlWindowBoxGet**

```
axlWindowBoxGet(  
)  
⇒ l_bBox
```

### **Description**

Returns the bounding box of the Allegro PCB Editor window currently visible to the user, in design units.

### **Arguments**

None.

### **Value Returned**

*l\_bBox*      bBox of the current Allegro PCB Editor window.

## **axlWindowBoxSet**

```
axlWindowBoxSet(  
    l_bBox  
)  
⇒ l_bBox/nil
```

### **Description**

Sets Allegro 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 Allegro 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.



#### *Caution*

***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.

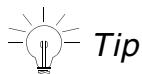
## axlPurgePadstacks

```
axlPurgePadstack (
  S_mode
  t/nil
)
⇒ x-cnt
```

### Description

Purges unused padstacks from the database in the area controlled by *S\_mode* symbol.

<b>S_mode symbol</b>	<b>2nd arg = t</b>	<b>2nd arg = nil</b>
'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 Allegro PCB Editor user interface.



For best results, first delete the unused padstacks from the database, then purge the via lists.

## Arguments

*S\_mode*      'padstacks or 'via.  
*option*      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 Allegro PCB Editor user interface behavior.

## **axlPurge3DModelMapDataInDesign**

axlPurge3DModelMapDataInDesign  
-> t/nil

### **Description**

Removes 3D model mapping data from the entire design.

### **Arguments**

None

### **Value Returned**

t	purge done
nil	not done

### **Example**

```
result = axlPurge3DModelMapDataInDesign()
```

## **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, Allegro 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/ash/ashshape.il

```
axlShapeAutoVoid(shapeDbid '(noRipThermals fragment))
```

## **axlShapeChangeDynamicType**

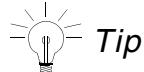
```
axlShapeChangeDynamicType (
    o_shapeId
    g_dynamic
    g_msgs
) -> o_dynShapeId/l_staticShapeId/nil
```

### **Description**

Swaps a connectivity shape from static to dynamic or the reverse. This offers the same functionality as the Allegro 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 Allegro PCB Editor converts the generated shapes (on ETCH) to static shapes not boundary shapes.
- Shapes converted to static maintain voids.
- Filled rectangles are supported on ETCH, converted to a shape.



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

<i>o_shapeId</i>	Dynamic shape id or static id.
<i>g_dynamic</i>	<i>t</i> makes the shape dynamic, <i>nil</i> makes the shape static.
<i>g_msgs</i>	<i>t</i> issue error messages if failed to convert; else be silent

## Value Returned

<i>nil</i>	Failure.
<i>o_dynShapeId</i>	<i>dbid</i> of the dynamic shape converted from static.
<i>l_staticShapeId</i>	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)
```

## **axIShapeDeleteVoids**

```
axIShapeAutoVoid(  
    o_shapeId/o_voidId/lo_voidid  
) -> t/nil
```

### **Description**

Lets you delete voids in a shape. Supports the following three forms of arguments:

- Shape that deletes all voids in that shape
- Delete single void
- Delete list of voids

Non-voids in list of voids options are silently ignored. You cannot delete the voids that are a part of auto-generated shapes.

If you are making a series of modifications to a shape, such as, deleting and adding voids or changing the shape boundary, then for best performance, it is recommended that you wrap your calls in [axIDBOpenShape](#) and [axIDBCreateCloseShape](#).

### **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**

<i>t</i>	Deletes voids.
<i>nil</i>	Error.

### **See Also**

[axIShapeAutoVoid](#), [axIDBOpenShape](#), [axIDBCreateCloseShape](#)

## Examples

See <cdsroot>/share pcb/examples/skill/axlcore/ashshape.il

Assuming you have shape `dbid (shapeId)`:

- Delete a single void

```
axlShapeDeleteVoids(car(p->voids))
```

- 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**

<code>o_shapeDbid</code>	dbid of a dynamic shape.
<code>g_force</code>	Force shape to update even if it is up to date.

### **Value Returned**

<code>x_ood</code>	If updating all returns count of all shapes that failed in updating. If single shape returns 0; update successful, 1 otherwise.
<code>nil</code>	Return if there is an error; <code>dbid</code> is not a dynamic shape.

### **Examples**

#### ■ Force update of a dynamic shape

```
axlShapeDynamicUpdate(shapeId, t) -> 0
```

#### ■ Update all shapes ood

```
axlShapeDynamicUpdate(nil nil) -> 0
```

## **axlShapeRaisePriority**

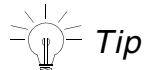
```
axlShapeRaisePriority(  
    o_shapeId  
) -> x_priority/nil
```

### **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. Allegro 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`).



*Tip*  
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**

<i>x_priority &gt; 0</i>	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)
```

## **axlShapeMerge**

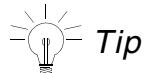
```
axlShapeMerge(  
    o_shapeId  
    lo_shapes  
    g_options/lg_options  
) -> o_dynShapeId/l_staticShapeId/nil
```

### **Description**

This merges shapes. Shapes must be overlapped without the fixed property to merge. All merging shapes (`lo_shapes`) must overlap the primary shape (`o_shapeId`).

Supports merging db types; shapes, rectangle and filled rectangles.

The resulting shape will take on the characteristics of the first shape. This includes shape type and properties. Any properties on the secondary shapes are lost.



*Tip*  
If changing 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> make shape dynamic, <code>nil</code> make static
<code>g_options</code>	Available options are: <code>check</code> - do not merge only perform checks for merging <code>'quiet</code> - do not output any messages

### **Value Returned**

- `nil`: indicates failure
- `o_dynShapeId`: the dbid of the dynamic shape that was converted from static
- `l_staticShapeId`: list of static shapes that was converted from a dynamic shape.

**See Also**

[axlShapeChangeDynamicType](#)

**Example**

ashOne is a shareware utility that allows user to select an object (See *<cdsroot>/share pcb/examples/skill/ash/ashshape.il*)

- Merge two shapes; interactively select two shapes that overlap

```
s1 = ashOne("shapes")
s2 = ashOne("shapes")
shapeResult = axlShapeMerge(s1 s2 'quiet)
```

## axlShovelItems

```
list  
axlShovelItems (  
    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()  
    axlShovelItems (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:

<b>shoveMode</b>	<b>Description</b>
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:

<b>cornerType</b>	<b>Description</b>
90	90 degree corners.
45	45 degree corners.
0	Any angle corners.

## Allegro SKILL Reference

### Interactive Edit Functions

---

*Gridded* is an integer as shown:

<b>gridded</b>	<b>Description</b>
0	Ignore grids (default)
1	Perform shoves on grid.

*Smooth* allows smoothing of shoved traces and is an integer as shown:

<b>smooth</b>	<b>Description</b>
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:

<b>oops</b>	<b>Description</b>
0	<i>Oops</i> off (default)
1	<i>Oops</i> if DRCs are left over.

*Samenet* tests for the same net 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.

<b>samenet</b>	<b>Description</b>
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.

## **axlSmoothDesign**

```
axlSmoothDesign(  
    lx_numPasses  
) -> x_change
```

### **Description**

Smooths the entire design. For good results on complicated designs, multiple passes are necessary. Since changes in one pass may open space that can be used in the next pass. Suggest 3 is a typical number although very complex designs can benefit from a higher number of passes. But the more passes the longer it will take.

### **Arguments**

lx\_numPasses      list of number of passes to perform

### **Value Returned**

x\_change, number of items changed

### **Example**

Smooth design using 3 passes

```
axlSmoothSetParams(list("45" "-1.0 "0" 10.0 0))  
res = axlSmoothDesign(list(3))
```

### **See Also**

[axlSmoothSetParams](#)

## **axlSmoothItems**

```
axlSmoothItems (
    lo_clineList
) ==> (x_list
```

### **Description**

Takes a list of dbids representing clines and/or cline segments and smooths them according to the parameters set using the [axlSmoothSetParams\(\)](#) function.

### **Arguments**

`lo_clineList`      List of dbids representing clines and/or cline segments to be smoothed.

### **Value Returned**

This function returns a list containing the number of clines that were changed by the smoothing process and the list of changed items. The format is as follows:

`(x_change (o_dbid1 o_dbid2 o_dbid3))`

Where `x_change` indicates the number of items changed, or `-1` if a user interrupt occurred.

If an error occurs, the function will return `nil`.

### **Example**

#### ■ Smooth a set of clines

```
clines = <list of ...>
axlSmoothSetParams(list("45" -1.0 "0" 10.0 0))
res = axlSmoothItems(clines)
```

### **See Also**

[axlSmoothSetParams](#)

## axlSmoothSetParams

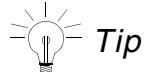
```
axlSmoothSetParams(  
    l_params  
) ==> t/nil
```

### Description

Sets the parameters used for smoothing the routes. All parameters must be supplied but a nil as a parameter option will leave the existing setting.

The smooth functionality is provided on an "as-is" basis. It works well on many designs but has the following restrictions:

- not differential pair aware.
- may have issues with electrically constrained nets



See [axlDBIgnoreFixed](#) if you want to temporary disable FIXED testing.

### Arguments

`l_params`      List containing the parameters, the list is of the following format.

```
(cornerType maxCornerLength padEntryRestriction minPadEntryLength  
    sortDirection)
```

## Allegro SKILL Reference

### Interactive Edit Functions

---

cornerType	Can be one of the following string values: 90 for 90 degree corners 45 for 45 degree corners 0 for any angle corners 1 for arc corners
maxCornerLength	This is an integer value indicating the maximum length of a bubble or jog in db units. A negative value indicates UNLIMITED.
padEntryRestriction	Can be one of the following string values: 2 Indicates that there are no restrictions 1 Indicates that all pad entry segments be fixed 0 Indicates that the entry segments for all rectangular pads be fixed
minPadEntryLength	This is an double value indicating the minimum length of fixed pad entry segments in user units. If a pad entry segment is longer than this length, it will be broken at or near that point so that smoothing can occur on that segment. A negative value indicates UNLIMITED. This value is not applicable if padEntryRestriction is "2".
sortDirection	This indicates how the clines are to be sorted before smoothing begins. This can be one of the following integer values: 0 No sorting. 1 Sort from the North. 2 Sort from the NorthEast. 3 Sort from the East. 4 Sort from the SouthEast. 5 Sort from the South 6 Sort from the SouthWest 7 Sort from the SouthWest 8 Sort from the NorthWest

## Value Returned

`t` if successful, `nil` if not.

## Example

- Set parameters

```
axlSmoothSetParams(list("45" -1.0 "0" 10.0 0))
```

- Update corner type to 90

```
axlSmoothSetParams(list("90" nil nil nil nil))
```

## See Also

[axlSmoothItems](#), [axlSmoothDesign](#), [axlDBIgnoreFixed](#)

## **axlStepDelete**

```
axlStepDelete(  
    g_type  
    t_name/o_dbid  
)  
==> t/nil
```

### **Description**

This removes a STEP assignment to a symbol or component definition.

### **Arguments**

<i>g_type</i>	Should be 'primary or 'alt (a nil is equiv to 'primary)
<i>t_name</i>	May be compdef or a symdef name. For a .dra file this should be nil
<i>o_dbid</i>	A symdef or compdef

### **Value Returned**

<i>t</i>	Deleted
nil	Not deleted

### **See Also**

[axlStepGet](#)

### **Examples**

- Delete STEP assignment on a symbol definition  

```
result = axlStepDelete(nil "LED_Y")
```
- Delete STEP assignment on a component definition  

```
result = axlStepDelete(nil "CAP")
```

## **axlStepSet**

```
axlStepSet(  
    g_mapType  
    t_name/o_dbid  
    t_stepFile  
    ll_mappingData  
)  
==> t/nil
```

### **Description**

Assigns a STEP file with a mapping or change to an existing STEP mapping to a symbol or component definition.

Offsets are specified in current design units. If STEP file uses different units, Allegro will eventually convert the offset values to the units of the STEP file.

Supports both .stp and .step file extensions.

### **Arguments**

<i>g_mapType</i>	Should be 'primary or 'alt (a nil is equiv to 'primary)
<i>t_name</i>	May be compdef or a symdef name
<i>o_dbid</i>	A symdef or compdef. For a .dra file this should be nil
<i>t_StepFile</i>	This is a name of the STEP file. If you have no directory component.  If this is nil and the object has the existing step file, then the mapping data provided is updated.
<i>ll_mappingData</i>	Can be nil if default mapping or list of symbol name/value pairs (see axlStepSet for permissible attributes)

### **Value Returned**

<i>t</i>	Mapping occurred
nil	Error

### **See Also**

[axlStepGet](#)

## Examples

- Assign a STEP model to a .dra design (no mappings)

```
result = axlStepSet('primary nil "led-L813_Y" nil)
```

- Assign a STEP model to a .dra design (no mappings)

```
map = '(rotation_z 3.1 rotation_y 2.0 rotation_x 1.0 offset_z  
6.3 offset_y 5.3 offset_x 4.2 color 9)  
result = axlStepSet('primary nil "led-L813_Y" map)
```

- Modify mappings from last example

```
map = '(rotation_z 30.1 rotation_y 20.0 rotation_x 10.0 offset_z  
6.3 offset_y 5.3 offset_x 4.2)  
result = axlStepSet('primary nil nil map)
```

- Assign a alternative STEP model to a symdef

```
map = '(rotation_z 10.1 rotation_y 20.0 )  
result = axlStepSet('alt "CAP10X16MM-RAD" "CAPC3225" map)
```

## **axlSymbolAttach**

```
axlSymbolAttach(  
    o_symInstDbid  
    o_dbid/lo_dbid  
)  
==> t/nil
```

### **Description**

Attaches an object or list of objects to symbol instance. For etch objects, provides the ability to associate pin escapes with a symbol.

Attach/detach rules:

- For detaching, object must be linked to the symbol instance provided. For attaching, the object can not be linked to any other symbol.
- If text appears of a REFDES class, it can't be linked or unlinked
- Linking or unlinking non-etch objects cause them to be deleted or duplicated if refresh symbol is run. Etch objects is more fully supported by refresh symbol.
- CLINES, LINES, SHAPES, RECTS, FRECTS and TEXT are supported with the layer limits noted below:
  - Items on BOUNDARY class items are NOT supported.
  - Shapes cannot be dynamic or generated from a dynamic shape.
  - Objects cannot be on the DRC class.



***Running refresh\_symbol may result in deletion of design level attachments.***

## Arguments

<i>o_symInstDbid</i>	symbol instance
<i>o_dbid</i>	dbid to assign to symbol
<i>lo_dbid</i>	list of dbid to assign to symbol

## Value Returned

<i>t</i>	was able to change object
<i>nil</i>	otherwise

## See Also

[axlSymbolDetach](#)

## Example

Examples use ashOne which is a shareware utility that allows user to select an object (see <cdsroot>/share pcb/examples/skill/ash-fxf/ashone.il)

■ Attach an object to a symbol:

```
symdbid = ashOne("SYMBOLS")
dbid = ashOne("NOALL")
ret = axlSymbolAttach(symdbid dbid)
```

## **axlSymbolDetach**

```
axlSymbolDetach(  
    o_symInstDbid  
    o_dbid/lo_dbid/g_mode  
)  
==> t/nil
```

### **Description**

Remove an object from a symbol instance. This function unlinks objects from the given symbol instance.

For pin escapes, two special modes are provided to detach all or most symbol etch from a given symbol instance.

It only unlinks an object from a symbol if it matches the provided symbol instance.

See [axlSymbolAttach](#) for the rules.



***Running refresh\_symbol result results in duplicate objects as the detached objects are loaded again.***

### **Arguments**

<i>o_symInstDbid</i>	symbol instance to modify.
<i>o_dbid</i>	dbid to unlink from symbol
<i>lo_dbid</i>	list of dbids to unlink from symbol
<i>g_mode</i>	special modes for unlinking all "etch" from symbol

- 'allEtch – deassign all etch from symbol
- 'allClineVia – design all etch except for shapes from symbol

## Value Returned

<i>t</i>	was able to change symbol
nil	Otherwise

## See Also

[axlSymbolAttach](#)

## Example

Examples use ashOne which is a shareware utility that allows user to select an object (see <cdsroot>/share pcb/examples/skill/ash-fxf/ashone.il)

- Typical method: To get the symdbid from the object:

- if etch

```
symdbid = dbid->symbolEtch
```

- if nonetch

```
symdbid = dbid->parent
symdbid = ashOne("SYMBOLS")
dbid = ashOne("NOALL")
ret = axlSymbolDetach(symdbid, dbid)
```

- Deassign all etch from symbol except shapes

```
symdbid = ashOne("SYMBOLS")
ret = axlSymbolDetach(symdbid, 'allClineVia)
```

## **axlAddTaper**

```
axlAddTaper(  
    o_dbid/lo_dbid  
)  
==> t/nil
```

### **Description**

Adds tapered trace. Tapered trace parameters are controlled from the Glossing "Pad and T" Parameter form.

### **Arguments**

*o\_dbid* dbid can either be a Path (CLINE) or line (segment).

### **Value Returned**

- *t*, returned when the function call is successful
- *nil*, indicates failure

## **axlTextOrientationCopy**

```
axlTextOrientationCopy(  
    o_textDbid  
    [orient]  
) -> orient/nil
```

### **Description**

This is a convenience function that updates a TextOrientation defstruct based upon a text dbid. This is typically used with [axlDBCreateText](#) or [axlDBChangeText](#).

### **Arguments**

o_textDbid	text dbid
orient	optional existing defstruct, if nil will create a new defstruct

### **Value Returned**

- orient, update TextOrientation defstruct
- nil, if there are error in the arguments

### **See Also**

[axlDBChangeText](#)

## axlTransformObject

```
axlTransformObject(  
    lo_dbid/o_dbid  
    ?move l_deltaPoint  
    ?mirror t/nil/'GEOMETRY  
    ?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 Allegro PCB Editor database object has a legal set of transforms (see [Table 5-1](#) on page 413). 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.

**Table 5-1 Supported Transforms**

OBJECT	MOVE	MIRROR	GEOMETRY	ROTATE	SPIN	ORIGIN (5)	NOTES
segments	X	X	X	X		box	
cline	X	X	X	X		box	
line	X	X	X	X		box	
symbol	X	X		X		xy	
shape	X	X	X	X		box	
text	X	X	X	X		xy	

## Allegro SKILL Reference

### Interactive Edit Functions

---

**Table 5-1 Supported Transforms, *continued***

OBJECT	MOVE	MIRROR	GEOMETRY	ROTATE	SPIN	ORIGIN (5)	NOTES
pin	X			X		xy	3, 4
via	X	X	X	X		xy	
rat_t	X			X		xy	
group	X	X	X	X		xy	7

### Notes

1. If object is not listed, then it is not supported.
2. If object has attached text, it also has the 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` column shows what rotate/mirror uses when operating on a single object without the origin option. For box, the `dbid` does not 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 the origin. For further discussion, see the note 9 on angles.
7. This API rejects objects whose owner is a symbol definition
8. The only groups that support a transform are user and module group types.
9. If mirror is `t` then we mirror in x-direction AND across subclasses. For example, if object is on ETCH/TOP it will be mirrored both in x and to layer ETCH/BOTTOM. If mirror is ``GEOMETRY` only a x-direction is done and the object remains on its layer.
10. 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 1_rotatePoint)`.

This point is then used as the rotation point.

## Cautions

- More objects may be added in the future. For example, voids.
- 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.
- If transforming multiple objects, enclose this operation in an `axlDBCloak` call.
- If transforming a segment, it will have a new owning path `dbid`.

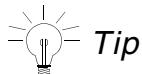
## Arguments

<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:

- An object can't be transformed (for example, a net)
- An object is fixed or a pin does not have an `UNFIX_PINS` property.
- Illegal option types used.
- Transformed object is outside of the database extents.



**Tip**

For better performance when transforming a group of objects, call this function with the object group instead of passing each *dbid* individually.

**See Also**

[axlDBCloak](#), [axlCopyObject](#)

**Examples**

*dbid* represents one database objects.

*ldbids* represents a list of database objects.

**Example 1**

```
axlTransformObject(ldbids, ?move '(100.0 0.0))
```

Moves a set of objects 100 database units vertically.

**Example 2**

```
axlTransformObject(dbid ?angle 45)
```

Rotates an object about its origin 45 degrees.

**Example 3**

```
axlTransformObject(dbid ?angle 45 ?origin 100:100)
```

Rotates an object about a rotation point.

## **axlZoneDelete**

```
axlZoneDelete(  
    t_zone/o_dbidZone  
) -> t/nil
```

### **Description**

Deletes a zone and its outline and associated shapes.

If symbols are placed on the surface layers of the zone, the zone is not deleted.

### **Arguments**

<i>t_zone</i>	Zone name
<i>o_dbidZone</i>	Dbid of zone

### **Value Returned**

<i>t</i>	If successful
<i>nil</i>	If failed

### **See Also**

[axlZoneCreate](#)

### **Examples**

#### ■ Delete a zone by name

```
result = axlZoneDelete("ZONE_1")
```

#### ■ Delete a zone by dbid

```
zone = axlZoneAccess('name "ZONE_1")  
when(zone axlZoneDelete(zone))
```

## axlZoneSet

```
axlZoneSet(  
    t_name/o_dbidZone  
    s_option  
    g_argument  
    ...  
)  
-> t/nil
```

### Description

Modifies characteristics of a zone. More than one set of option/arguments can be provided. They are:

'name t_newName	Rename a zone
'notes t_notes	Add or update the note  If <i>t_notes</i> is nil deletes the note
'stackup t_stackup	Updates the stackup associated with zone stackup must currently exist and is rejected if surface symbols placed in the zone
'room t_roomName	Assign room name to zone. Room with same name must not exist in the design. Creates a room shape for BOTH_ROOMS.  If <i>t_roomName</i> is nil deletes room name.
'constraint t_constraintName	Assign constraint region.  Region must already exist in design and creates a shape on subclass "ALL".  If <i>t_constraintName</i> is nil deletes constraint area.

### Arguments

t_name	Name of zone
o_dbidOutline	Outline (shape) on class==RIGID_FLEX and subclass==RIGFLEX_ZONE_OUTLINE

### Value Returned

t	If successful
---	---------------

nil                    If failed

## See Also

[axlZoneCreate](#), [axlXSectionCreate](#)

## Examples

Setup zone including rename; associating a stackup and room

```
ret = axlZoneCreate("ZONE_1" 'stackup "FLEX_1" 'room "ROOM_1")
```

## Padstack Access Functions

This section lists padstack access functions.

### 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*.

#### *Defstructs used to create padstack*

Drill (*r\_drill*) use make\_axlPadStackDrill. Elements are:

usage	(string) padstack usage. See <a href="#">axlPadstackUsageTypes()</a> . If this parameter is not provided, code determines type based on padstack characteristics.
fixed	( <i>t/nil</i> ) internal fixed flag
spanLockCount	( <i>t/nil</i> ) Default value is <i>nil</i> . If <i>t</i> , padstack does not expand or contract if layers are added or deleted. Default it expands/contracts.
uvia	( <i>t/nil</i> ) if padstack is of type <code>bbvia</code> set as sub-type micro-via. Superseded by usage type.
keepout	( <i>t/nil</i> ). Obsolete for 17.0 and above. Use new 'KEEPOUT pad layer type
pluralVia	( <i>t/nil</i> ). Create padstack as a plural via (multi-net). Only through hole padstacks can have this setting.
holeType	(symbol) the hole type. Allowed symbols are 'CIRCLE_DRILL, 'SQUARE_DRILL, 'OVAL_SLOT, 'RECTANGLE_SLOT. Defaults to CIRCLE_DRILL if drill diameter is provided.

## Allegro SKILL Reference

### Interactive Edit Functions

---

<i>plating</i>	(symbol) plate status of drill hole Symbols are: 'PLATED, 'NON_PLATED or nil
<i>drillDiameter</i>	(float) drill hole finished diameter.
<i>drillToolSize</i>	(string) Drill tool size, which is used as an identifier. Default is blank.
<i>slotSize</i>	((f_width f_height)) size of slot hole. Use this instead of drillDiameter for SLOT types.  The f_width or "Y size" is the drill size.
	The f_height or "X size" or drill travel.
<i>holeTolerance</i>	( (f_pos f_neg) ) +/- hole tolerance
	When a slot defines the X tolerance or route tolerance
<i>holeToleranceY</i>	( (f_pos f_neg) ) +/- hole tolerance
	Applies only to slots and defines the drill size tolerance (Y tolerance). This is the slot width.
<i>offset</i>	( (f_x f_y) ) drill hole offset
<i>multiDrillData</i>	list for multiple drill data which is:  ( (x_num_rows x_um_columns f_clearance_x [f_clearance_y ["staggered"]]) )  data type is (int int float [float])
<i>drillNonStandard</i>	(symbol) non-standard drill hole.  Supported symbols are:  'LASER_DRILL, 'PLASMA_DRILL, 'PUNCH_DRILL, 'PHOTO_DRILL, 'COND_INK_DRILL, 'WET_DRY_DRILL, 'OTHER_DRILL.

## Allegro SKILL Reference

### Interactive Edit Functions

---

<i>figure</i>	(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.
<i>figureSize</i>	( (f_width f_height) ) size of drill figure
<i>drillChar</i>	(string) drill characters. Maximum 3 characters.
<i>holeCounterType</i>	(string) Counter hole type: "bore", "sink" or nil
<i>holeCounterAngle</i>	(integer) Applies to Counter sink and indicates an angle between 1 and 90 degrees.
<i>holeCounterDiameter</i>	(float) Counter hole diameter in design units
<i>holeCounterDepth</i>	(float) Applies to Counter bore and indicates depth of the bore. Accuracy is maintained at a higher level then the current design.
<i>holeCounterTolerance</i>	( (f_pos f_neg) ) +/- counter hole tolerance.
<i>backdrillDiameter</i>	(float) diameter in design units of backdrill finished hole
<i>backdrillFigureName</i>	(symbol) the drill figure (see figure for allowed types)
<i>backdrillFigureChar</i>	(string) backdrill characters (3 max)
<i>backdrillFigureWidth</i>	(float) width of backdrill figure in design units
<i>backdrillFigureHeight</i>	(float) height of backdrill figure in design units

Pad (*l\_pad*) structure (up to 4 for each layer indicated by attribute type)

<i>layer</i>	(string) etch layer name (for example, "TOP") or "DEFAULT_INTERNAL" if you want one pad layer to map to all internal layers between the top and bottom of the padstack.
<i>type</i>	(symbol) pad type. Allowed symbols are: KEEPOUT, ANTIPAD, THERMAL or REGULAR. nil is treated as REGULAR.

## Allegro SKILL Reference

### Interactive Edit Functions

---

*figure*

(symbol) the pad figure

For allowed symbols, see axlPadFigureTypes API.

If NULL the figureSize is checked and automatically assigns 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.

For shape symbol use the name of the ssm (minus extension and path) to figure.

For example, if you have a shape symbol called "myshape" then it would be '?figure "myshape"'.

For an Anti-pad shape (fsm) assign the symbol 'FLASH and assign the fsm file (minus extension and path) to the flash name. For example, symbol "myflash" would be:

'?figure 'FLASH ?flash "myflash" '

For either a shape or flash, the symbol must be located through PADPATH. Also the ?figureSize attribute must be the extents of the symbol or larger.

*flash*

(string) the pad aperture flash name.

Reference a flash shape symbol name or nil for no flash (fsm file).

*figureSize*

( (f\_width f\_height) ) height and width of the figure. For a circle, you only need to assign diameter to either height/width, the other can be 0.

*offset*

( (f\_x f\_y) ) offset from the padstack origin

*sides*

Octagon pad only: Number of sides. Ranges between 6 and 64 as an even integer.

*inside*

Donut pad only: inside dimension as a floating point user unit

*radius*

ROUNDED\_RECTANGLE and CHAMFERED\_RECTANGLE pad only, corner radius as a floating point user unit.

*corners*

ROUNDED\_RECTANGLE and CHAMFERED\_RECTANGLE pad only, a dash separated string indicating corners chamfered:

UR - upper right

UL - upper left

LR - lower right

LL - lower left

*nil* if pad type does not support

Example: "UR-LR" means chamfer Upper Right and Lower Left corners.

## PROGRAMMING NOTES:

- N\_SIDED\_POLY – figureSize uses the smaller of width and height and this maps to diameter. Requires sides and if not present pad is turned into a circle. Sides must be an even number between 6 and 64. If sides is 0 a CIRCLE pad is created.
- ROUNDED\_RECTANGLE/CHAMFERED\_RECTANGLE – Requires radius and corners string. Radius cannot be larger than half the smaller of height or width. If radius is 0 or corners string does not match any corners then a RECTANGLE pad is created.
- DONUT – figureSize uses the smaller of width and height and this maps to outside diameter. Requires an inside diameter that must be smaller then the outside diameter. If inside diameter is zero then a CIRCLE pad is created
- To create an adjacent layer route keep-out pad, use the following options for make\_axlPadStackPad
  - ?layer 'adjacent
  - ?type 'KEEPOUT
- To create Overlay pad layers
  - use strings for ?layer "COVERLAY\_TOP" or "COVERLAY\_BOTTOM"
  - ?type is ignored
- To create backdrill pad layers use:
  - ?layer 'backdrillStart
    - ?type "REGULAR" – larger pad for drill start layer (positive)

## Allegro SKILL Reference

### Interactive Edit Functions

---

- ?type "ANTI" – larger pad for drill start layer (negative layers)
- ?layer 'backdrillClearance
  - ?type "ANTI" – pad size for internal backdrill layers for negative layers
  - ?type "KEEPOUT" – generate a KEEPOUT for positive layers for backdrill layers.
  - ?layer 'backdrillSoldermask – soldermask pad to substitute for drill
- Shapes and flash assignment symbol rules:
  - 'REGULAR – shape symbols (flash symbol name can be set for legacy Gerber support)
  - 'THERMAL – flash symbols
  - 'ANTIPAD - shape symbol (flash symbol name can be set for legacy Gerber support)
  - 'KEEPOUT - shape symbol
  - Mask layers - either flash or shape symbols
- A shape symbol can contain one shape with NO voids.
- A flash symbol supports multiple shapes with voids.
- User mask layer support requires a call to axIPadUserMaskLayers with the 'create option to insure a user mask layer exists in the design. If you know that the user mask layer already exists, you do not need to make the call.
- Backdrill data can be seeded by the global design parameters. Padstacks with their backdrill data obtained this way show these values but they are not saved to the disk padstack file (.pad).

## Arguments

<i>t_name</i>	Padstack name.
<i>r_drill</i>	Drill hole data for the padstack.  <b>Note:</b> As with all SKILL defstructs, use the constructor function, <code>make_axlPadStackDrill</code> , to create instances of <code>axlPadStackDrill</code> . See the example.
<i>l_pad</i>	Pad definition data for the padstack.  <b>Note:</b> As with all SKILL defstructs, use the constructor function <code>make_axlPadStackPad</code> to create instances of <code>axlPadStackPad</code> . See the example.
<i>g_nocheck</i>	Optional. <i>t</i> disables checks of the padstack definition. <i>nil</i> executes the following checks of the padstack definition: <ul style="list-style-type: none"><li>- Contiguous pad definitions</li><li>- Anti-pad / thermal-relief pad definitions</li><li>- Existence of two pads with a drilled hole</li><li>- A drilled hole with the existence of two pads</li></ul>

## Value Returned

<i>l_result</i>	<i>dbid</i> of the padstack created.
<i>nil</i>	Nothing is created.

## See Also

[axlPadstackEdit](#), [axlLoadPadstack](#), [axlDBCopyPadstack](#), [axlPadstackToDisk](#), [axlDBGetPad](#), [axlPadUserMaskLayers](#), [axlPadFigureTypes](#), [axlPadstackUsageTypes](#), [axlPadDbidDoc](#)

## Examples

### ■ Surface Mount Padstack Example

Adds a surface mount padstack having a 25 by 60 rectangular pad.

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

### ■ Additional examples in:

## **Allegro SKILL Reference**

### Interactive Edit Functions

---

<cdsroot>/share pcb/examples/skill/dbcreate/pad.il

## **axlPadFigureTypes**

```
axlPadFigureTypes()  
    => lt_names
```

### **Description**

Returns list of strings of supported pad figure types. Not all pad types and layers may support a figure type. This is a documentation support function.

### **Arguments**

None

### **Value Returned**

<i>lt_names</i>	List of permitted pad figure types
-----------------	------------------------------------

### **Examples**

```
axlPadFigureTypes()
```

## axlPadstackEdit

```
axlPadstackEdit(  
    nil  
    nil  
)  
==> l_attributes  
  
axlPadstackEdit(  
    o_dbidPadstack/t_Padstack  
    s_name  
    g_value  
)  
==> t/nil  
  
axlPadstackEdit(  
    o_dbidPadstack/t_Padstack  
    [[s_name g_value] .... ]  
)  
==> t/nil
```

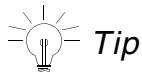
### Description

Inquire and set display options. Edits global settings of an existing padstack.

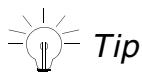
This edits the padstack definition, this means that any changes made applies to all instances of the padstacks (pins and vias) in the design.

Supports the following modes:

- If first two arguments are `nil`, command returns a list of all editable attributes.
- If padstack, attribute, and new value are provided, changes one attribute of padstack.
- If padstack, and a list of attributes with new values are provided change all the items specified in the padstack. This is the most efficient method for changing multiple items on a single padstack.



*Tip*  
Order is important, so if you are changing a CIRCULAR drill to a slot then you must provide the holeType followed by drillSizeWidth and drillSizeHeight.



*Tip*  
For best performance if changing multiple items in a single padstack use the list mode to change all items in one call.

## **Allegro SKILL Reference**

### Interactive Edit Functions

---

Currently only global padstack settings are supported. Editing pad layer characteristics is not allowed. Certain changes set DRC out of date and dynamic shapes out of date. Attributes currently supported (all Equivalent items are field names in pad\_designer):

**Table 5-2 Supported Controls**

Name	Value	Description	Equivalent	Side Effects
usage	<p>symbol or string with possible values as:</p> <p>Types, the call axlPadstackUsa geTypes() returns available usage types:</p> <ul style="list-style-type: none"> <li>'Smd</li> <li>'Through</li> <li>'Through_pin</li> <li>'Surface_pin</li> <li>'Bbvia</li> <li>'Through_via</li> <li>'uvia</li> <li>'Bond_finger</li> <li>'Die_pad</li> <li>'Fiducial</li> <li>'Mechanical_hole</li> <li>'Mounting_hole</li> <li>'Slot</li> </ul>	<p>Changes classification of padstacks. In some cases the use of the padstack may override the actual padstack usage type.</p> <p><b>Note:</b> Smd and Surface_pin are identical but future releases will check that Surface_pin is only used for pins. Similarly, Through, Through_via and Through_pin are the same but future releases will support a check.</p> <p><b>Transition rules</b> (types can be switched among themselves):</p> <ul style="list-style-type: none"> <li><i>Single layer:</i> Smd, Surface_pin, Die_pad, Bond_finger, Fiducial</li> <li><i>Multi layer:</i> Through, Through_pin, Through_via, Bbvia, uvia, Mechanical_hole, Mounting_hole, Slot</li> </ul> <p><b>Note:</b> Future releases may tighten the transition rules.</p>	<p>Padstack Usage type or 'usage attribute in the padstack dbid</p>	DRC is out of date and dynamic shapes are disabled

**Allegro SKILL Reference**  
Interactive Edit Functions

---

**Table 5-2 Supported Controls, *continued***

Name	Value	Description	Equivalent	Side Effects
drillDiameter	dbrep	Changes the drill diameter. Value must be a positive number and holeType must not be a slot. For multi-drill applies to all drills.	Drill diameter field	DRC is out of date and dynamic shapes are disabled
drillSizeWidth	dbrep	Changes the slot size width (x direction). Value must be a positive number and holeType must be a slot type. Usually done in association with drillSizeHeight.	Slot size X	DRC is out of date and dynamic shapes are disabled.
drillSizeHeight	dbrep	Changes the slot size width (y direction). Value must be a positive number and holeType must be a slot type. Usually done in association with drillSizeWidth.	Slot size Y	DRC is out of date and dynamic shapes are disabled.
drillToolSize	string	Sets the Drill tool size name. Drill tool size Not applicable if padstack does not have a drill.	Drill tool size	None

## Allegro SKILL Reference

### Interactive Edit Functions

---

**Table 5-2 Supported Controls, *continued***

Name	Value	Description	Equivalent	Side Effects
holeType	string	<p>One of "CIRCLE_DRILL", "SQUARE_DRILL", "OVAL_SLOT" or "RECTANGLE_SLOT". Changes hole type of padstack. If changing fundamental types:</p> <ul style="list-style-type: none"> <li>■ slot to drill; drillDiameter inherits drillSizeWidth.</li> <li>■ drill to slot; both drillSizeWidth and drillSizeHeight inherit drillDiameter</li> </ul> <p>For slots drillFigureName takes on figure for slot type selected and its width and height are the same as the slot width and height.</p>	Hole type	DRC is out of date and dynamic shapes are disabled.
drillNonStandard	string	Changes type of non-standard drill. Hole type must be a circular drill. Types are "LASER_DRILL", "PLASMA_DRILL", "PUNCH_DRILL", "PHOTO_DRILL", "COND_INK_DRILL", "WET-DRY_DRILL", and "OTHER_DRILL". Use nil if you want to unset this field.	Non-standard drill	None
drillOffset	point/dbrep	Changes the drill offset. Must be a xy point or a single dbrep, which applies to both, x and y.	Offset X and Y	DRC is out of date and dynamic shapes are disabled.

## Allegro SKILL Reference

### Interactive Edit Functions

---

**Table 5-2 Supported Controls, *continued***

Name	Value	Description	Equivalent	Side Effects
holeTolerance	point/dbrep	Changes the hole tolerance. Point values are taken to be a list of (+tolerance - tolerance) or a single dbrep which applies to both + and - . Both must be a positive number.	Tolerance + and -	None
holeToleranceY	point/dbrep	Changes the slot Y tolerance. Point values are taken to be a list of (+tolerance -tolerance) or a single dbrep which applies to both + and -. Both must be a positive number.	Y Tolerance + and - for slots	None
plating	string	Sets the plating type, must be one of "NON_PLATED", "OPTIONAL", or "PLATED".	Plating	None
pluralVia	t/nil	Sets padstack to have a plural via	Plural Via	Can only be set for through hole padstacks. Padstack cannot be in use in the design
drillChar	string	Sets the drill characters. A maximum string of 3 is supported. Longer strings are truncated. Use a nil to remove the string.	Characters	None

## Allegro SKILL Reference

### Interactive Edit Functions

---

**Table 5-2 Supported Controls, *continued***

Name	Value	Description	Equivalent	Side Effects
drillFigureName	string	Sets the drill figure type. Not supported for slots. Value must be a string that matches one of the drop-down items in the pad_designer "Figure" field.	Figure	None
drillFigureHeight	dbrep	Changes the figure height, value must be a positive number. Option not available for slots. Used in conjunction with drillFigureWidth.	Height (under Drill/ Slot symbol)	None
drillFigureWidth	dbrep	Changes the figure width, value must be a positive number. This is not available for slots. Used in conjunction with drillFigureHeight.	Width (under Drill/ Slot symbol)	None
backdrillDiameter	float	Changes the backdrill diameter. Value must be a positive number and holeType must be a regular hole.	Secondary Drill tab	None
backdrillFigureCh ar	string	Sets the drill characters. A maximum string of 3 is supported.  Longer strings are truncated. Use a nil to remove the string.	Secondary Drill tab	None
backdrillFigureNa me	string	Sets the backdrill figure type. Value must be a string that matches one of the drop-down items in the pad_designer "Figure" field.	Secondary Drill tab	None

## Allegro SKILL Reference

### Interactive Edit Functions

---

**Table 5-2 Supported Controls, *continued***

Name	Value	Description	Equivalent	Side Effects
backdrillFigureHeight	dbrep	Changes the backdrill figure height, value must be a positive number.  Used in conjunction with backdrillFigureWidth.	Secondary Drill tab	None
backdrillFigureWidth	dbrepdth	Changes the backdrill figure width, value must be a positive number.  Used in conjunction with backdrillFigureHeight.	Secondary Drill tab	None
spanLockCount	t/nil	Prevents non-through hole padstacks from expanding (or contracting) when layers are added (or deleted). Default is for these padstacks to maintain their original start/stop layers.	Lock layer span	padstacks with this options may create disconnects when new layers are added
uvia	t/nil	Sets the type of via to bbvia (nil) or micro via (t).  Padstack must be a bbvia	This is kept for backwards compatibility.  Use the 'type' option.	DRC is out of date and dynamic shapes are disabled.
keepout	nil	Obsolete in 17.0.  Use new route keepout pad layer type.		none

## Arguments

<i>o_dbidPadstack</i>	dbid of a padstack (note VIA and PIN dbids are not supported)
<i>t_Padstack</i>	Name of padstack
<i>s_name</i>	Symbol name of attribute to change
<i>g_value</i>	New value
<i>[ [s_name g_value]</i> <i>.. ]</i>	list of name/value pairs

## Value Returned

- *ls\_names* - If name is nil then returns a list of all controls.
- *t/nil* - if t successful in updating padstack, nil an error

## Examples

Finds a padstack using the ashOne share-ware SKILL. Note that selection will return a pin or via and you must get the padstack from the definition attribute.

```
p = ashOne()  
padstack = p->definition  
■ Set drill characters  
ret = axlPadstackEdit(padstack 'drillChar "abc")  
or its equivalent  
ret = axlPadstackEdit(padstack '((drillChar "abc")))
```

- Set tolerance  
ret = axlPadstackEdit(padstack 'holeTolerance '(1.2 1.3))
- Set tolerance same for + and -  
ret = axlPadstackEdit(padstack 'holeTolerance 1.5)

- Set drill symbol data  
data = '((drillFigureName "RECTANGLE") (drillFigureHeight 20)  
(drillFigureWidth 10) (drillChar A))  
ret = axlPadstackEdit(padstack data)

- Get list of all editable padstack parameters  
lst = axlPadstackEdit(nil nil)

## **Allegro SKILL Reference**

### Interactive Edit Functions

---

#### **See Also**

[axIDBCreatePadStack](#), [axILoadPadstack](#), [axIDBCopyPadstack](#), [axIReplacePadstack](#)

## **axlPadstackSetType**

```
axlPadstackSetType(  
    o_padstack/t_padstack  
    g_uviaBbvia  
) -> t/nil
```

```
axlPadstackSetType(  
    o_padstack/t_padstack  
    g_type  
    g_value  
) -> t/nil
```

### **Description**

Changes a padstack type. In its 2 argument mode is the same as:

```
axlPadstackSetType(padstack 'type g_uviaBbvia)
```

Permits changing the type of via.

'type

Changes a bbvia padstack to a micro via and vice versa. Uvia types can be managed separately in the constraints system. This has no effect if the padstack is used with Pins. Values are 'bbvia or 'uvia.

'keepout

Obsolete in 17.2. Create a keepout pad layer type of KEEPOUT.

Marks DRC out-of-date if successful.

## Arguments

<i>o_padstack</i>	padstack dbid
<i>t_padstack</i>	padstack name
<i>g_type</i>	mode (either 'type or 'keepout)
<i>g_value</i>	appropriate setting (see above)

## Value Returned

<i>t</i>	change successful
<i>nil</i>	failed. Not a padstack, padstack not in database, type not recognized or padstack not a bvia or uvia.

## Examples

Change padstack named VIA to a micro via

```
axlPadstackSetType("VIA" 'type 'uvia)
```

## See Also

[axlDBCreatePadStack](#), [axlPadstackEdit](#)

## **axlPadstackUsageTypes**

```
axlPadstackUsageTypes()  
    => lt_names
```

### **Description**

Returns list of strings of supported padstack usage types. This is a documentation support function.

### **Arguments**

None

### **Value Returned**

- List of strings

### **Examples**

```
axlPadstackUsageTypes()
```

## **axIPadUserMaskLayers**

```
axIPadUserMaskLayers (
    ['max]
) =>lt_names/x_cnt

axIPadUserMaskLayers (
    'create
    t_user_mask_layer
) =>t/nil
```

### **Description**

Supports following modes:

- if argument returns list of user mask layers in design
- 'max symbol option reports the maximum number that we support.
- 'create adds a new user mask layer; requires a name

User mask naming:

If you use \_TOP and \_BOTTOM suffixes, then you will get mirror support. This means if you mirror a VIA with a user mask defined on xxx\_TOP then it will be mirrored to the bottom. See axIDBControl('mirrorUserMask). With the mirror mask option enabled, the opposite side mask is auto-created.

## Arguments

'max	Optional. If provided, returns the maximum number of mask layers supported.
'create	Create a new user mask layer, requires t_user_mask_layer
t_user_mask_layer	Name of user mask layer

## Value Returned

lt_names	List of user mask names
x_cnt	Maximum number permitted to be defined
t/nil	In create mode, returns t if layer was created or exists, nil if failed to create. Failure can be due to:

- exhausted number of user mask layers available
- name is illegal

## See Also

[axIDBCreatePadStack](#), [axIDBControl](#)

## Examples

### 1. Typical use to get all layers defined

```
axlPadUserMaskLayers()
```

### 2. Return max that can be defined

```
axlPadUserMaskLayers('max') -> 32
```

## **Allegro SKILL Reference**

### Interactive Edit Functions

---

---

## **Database Read Functions**

---

### **AXL-SKILL Database Read Functions**

The chapter describes the AXL-SKILL functions that read the Allegro PCB Editor database.

## **axlAltSymbolList**

```
axlAltSymbolList(  
    t_name/o_dbid  
    g_layer  
) lt_symbols/nil
```

### **Description**

This queries the provided object and returns a list of alternative symbol names.

### **Arguments**

<i>t_name</i>	Component definition or a refdes name
<i>o_dbid</i>	a symbol instance, component instance or compdef dbid
<i>g_layer</i>	'top', 'bottom', or 'internal'

### **Value Returns**

list of alternate symbols for the layer provided

nil, in case of error or if the symbol does not have a alternate symbol set

### **Examples**

```
strings = axlAltSymbolList("U1" top)
```

### **See Also**

[axlAltSymbolReplace](#), [axlAltSymbolOK](#), ALT\_SYMBOL property

## **axlAltSymbolOK**

```
axlAltSymbolOK(  
    t_name/o_dbid  
    g_layer  
    t_symbol  
    l_dbpoint  
) => t/nil
```

### **Description**

This verifies that symbol is legal for component. Must be in the ALT\_SYMBOL list with the correct layer.

### **Arguments**

<i>t_name</i>	may be compdef or a refdes name
<i>o_dbid</i>	a symbol instance, component instance or component definition dbid
<i>g_layer</i>	'top, 'bottom, or 'internal
<i>t_symbol</i>	name of symbol
<i>l_dbpoint</i>	'(x y) location on the board (optional)

### **Value Returns**

<i>t</i>	is legal
<i>nil</i>	error or symbol is not legal for component

### **Examples**

```
result = axlAltSymbolOK("R1" 'top "res400")
result = axlAltSymbolOK("R1" 'top "res400" '(6400 3300))
```

### **See Also**

[axlAltSymbolList](#), [axlAltSymbolReplace](#)

## **axlAltSymbolReplace**

```
axlAltSymbolReplace(  
    t_name/o_dbid  
    t_symbol  
) => t/nil
```

### **Description**

This replaces a PLACED component with one of its allowed replacements (ALT\_SYMBOL). To be successful the following must be true:

- ❑ The symbol must already be placed
- ❑ The provided symbol must be a legal alternative for the layer where the symbol is placed
- ❑ The replacement symbol, any of its padstacks and any shape or flash symbols referenced by the padstacks must be found and loaded into the design, which this API will attempt to do.

**Note:** Text properties on the symbol instance, and attached text re-positioning are preserved.

### **Arguments**

<i>t_name</i>	may be compdef or a refdes name
<i>o_dbid</i>	a symbol instance, component instance or component definition dbid
<i>t_symbol</i>	name of replacement symbol

### **Value Returns**

<i>t</i>	replacement done successfully
<i>nil</i>	error or cannot replace

### **Examples**

```
result = axlAltSymbolReplace("R1" "res400")
```

## **Allegro SKILL Reference**

### Database Read Functions

---

#### **See Also**

[axlAltSymbolList](#), [axlAltSymbolOK](#)

## axlBackdrillGet

```
axlBackdrillGet(  
    o_dbidPinOrVia  
) -> lt_backdrillData/nil  
  
axlBackdrillGet(  
    'status  
) -> g_status
```

### Description

In one mode, it returns status of design's backdrilling. In other mode, when a pin or a via is provided, the command returns the backdrilling result on that pin or via.

**Note:** Symbols do not support backdrilling.

### Arguments

<i>o_dbidPinOrVia</i>	query for backdrill on pin or via
<i>status</i> '	status of backdrill on design

### Value Returns

<i>nil</i>	error or pin/via is not backdrilled
<i>g_status</i>	returns status about design's backdrill
<i>nil</i>	error
'notStarted	backdrill not performed on design
't	backdrill done and is up to date
'odd	backdrill done but changes make it out-of-date
<i>lt_backdrillData</i>	A disembodied property list having (all non strings are in user units). Refer to <a href="#">Property List</a>

**Table 6-1 Property List**

---

<i>holeSize</i>	finished hole size
<i>backdrillSize</i>	backdrill hole size

## Allegro SKILL Reference

### Database Read Functions

---

<i>electricalStub</i>	max electrical stub (from the BACKDRILL_MAX_PTH_STUB property on net)
<i>mfgStub</i>	manufacturing stub from backdrill parameter dialog

If pin or via is backdrilled from top, topStartLayer is non-nil and the following attributes are set:

<i>topStartLayer</i>	start layer of drill (string)
<i>topMustCutLayer</i>	must cut layer (string)
<i>topMustNotCutLayer</i>	must NOT cut layer (string)
<i>topDrillDepth</i>	depth of backdrill from topStartLayer to immediately before topMustNotCutLayer
<i>topRemainingStub</i>	remaining stub after backdrill. This is from topRemainingStub to first layer with a connection (min value is manufacturing stub length)

If pin/via is backdrilled from bottom, botStartLayer is non-nil and the following attributes are set.

<i>botStartLayer</i>	start layer of drill (string)
<i>botMustCutLayer</i>	must cut layer (string)
<i>botMustNotCutLayer</i>	must NOT cut layer (string)
<i>botDrillDepth</i>	depth of backdrill from botStartLayer to immediately before botMustNotCutLayer
<i>botRemainingStub</i>	remaining stub after backdrill. This is from botRemainingStub to first layer with a connection (min value is manufacturing stub length)

---

## Examples

### ■ Get status

```
axlViaZLength('status)
```

### ■ Get info on a pin or via

```
; ashOne is a selection utility found at
;   <cdsroot>/pcb/examples/skill/ash-fxf/ashone.il
```

## **Allegro SKILL Reference**

### Database Read Functions

---

```
; select a pin or via  
  
pinvia = ashOne('("PINS" "VIAS"))  
backdrill = axlBackdrillGet(pinvia)
```

#### **See Also**

[axlViaZLength](#)

## 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:** You cannot edit the root design object. AXL-SKILL edit commands ignore this *dbid*.

This also allows access to lists of many types of dbid in the design, such as nets, components, and so on.



If you need to access all dbids if the same type (all nets or all components), it is more efficient to use the appropriate attribute of this dbid than to use the Selection APIs to query all elements.

To flatten attributes in lists, use the "*~>*" reference. For example, to get names of all nets in the design do:

```
netNames = axlDBGetDesign() ->nets~>name
```

### 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:

## **Allegro SKILL Reference**

### Database Read Functions

---

1. From the Allegro 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"/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" or  
"Nonplated"

Drillplating name.

nil

Incorrect padstack name, or other error occurred.

## axlIsDBIDType

```
axlIsDBIDType(  
    g_dbid  
)  
⇒ t/nil
```

### Description

Determines if *g\_dbid* is an Allegro PCB Editor database *dbid*. Returns *t* if so and *nil* otherwise.

### Arguments

*g\_dbid* Variable to be checked whether a *dbid* or not.

### Value Returned

*t* *g\_dbid* is a true Allegro PCB Editor *dbid*.

*nil* *g\_dbid* is not a true Allegro PCB Editor *dbid*.

### Example

Defines a function based on `axlIsDBIDType` to tell whether a symbol is an Allegro PCB Editor *dbid* or not. Then creates an *r\_path* (which is not an Allegro PCB Editor *dbid*, because *paths* are only temporary building structures) and uses the *r\_path* to create an Allegro PCB Editor line (which is an Allegro 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:

## **Allegro SKILL Reference**

### Database Read Functions

---

"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*. If [axlDBGetDesign](#) is used to retrieve the dbid, the function returns all text attached to root design.

### **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.

## **Allegro SKILL Reference**

### Database Read Functions

---

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*.

#### **Note:**

smd pads do not have a default internal layer.

To obtain the adjacent layer keepout pad: axlDBGetPad(dbid 'adjacent "KEEPOUT")

## Allegro SKILL Reference

### Database Read Functions

---

#### Arguments

*o\_dbid*      *dbid* of the pin, via, or a padstack definition.

*t\_layer*      String or symbol of a layer desired.

Format for regular layers:

- "ETCH/TOP"
- "TOP"
- Mask layers must use class of pin or via class:
  - "PIN/SOLDERMASK\_TOP"
- Coverlay pads specify by "COVERLAY\_TOP" or "COVERLAY\_BOTTOM"

Special layers:

- 'internal - default internal layer (not present on SMD padstacks)
- 'composite - sum of all the layers (worse case) this option ignores the *t\_type* argument
- 'adjacent – adjacent layer keepout option
- 'backdrillStart
  - type="REGULAR" – larger pad for drill start layer (positive)
  - type="ANTI" – larger pad for drill start layer (negative layers)
- 'backdrillClearance
  - type="ANTI" – pad size for internal backdrill layers for negative layers
  - type="KEEPOUT" – generate a KEEPOUT for positive layers for backdrill layers.
- 'backdrillSoldermask – soldermask pad to substitute for drill start layer

*t\_type*      Type of pad to retrieve: "REGULAR", "ANTI", "THERMAL", or "KEEPOUT".

## 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

- Lets the user pick any pin or via and shows the *figureName* attribute of the selected pad.

```
(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)))
```

To fetch default internal regular pad:

```
r = axlDBGetPad(car(axlGetSelSet()) 'internal "REGULAR")
```

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
```

## **axlDBGetPropDict**

```
axlDBGetPropDict(  
    S_filter/nil  
) -> lt_propNames
```

### **Description**

Returns a list of property definitions in the current design. Several trivial filters are defined:

- `nil` – visible Allegro and user-defined properties
- `allegro` – visible Allegro properties. Product tiering results in filtering of these properties.
- `user` – visible user-defined properties.
- `invisible` – Invisible Allegro properties, these are application internal properties.

### **Arguments**

<code>S_filter</code>	Symbol or string requesting a list of property types
-----------------------	--

### **Value Returned**

<code>lt_propNames</code>	List of names (unsorted) or <code>nil</code>
---------------------------	--

### **Examples**

```
axlDBGetPropDict('user) -> list of names
```

### **See Also**

[axlDBGetPropDictEntry](#)

## **axIDBGetPropDictEntry**

```
axIDBGetPropDictEntry(  
    t_name  
)  
⇒ o_propDictEntry/nil  
  
axIDBGetPropDictEntry(  
    nil  
)  
==> lt_validObjects
```

### **Description**

Gets the property dictionary entry for the property name given by the string *t\_name*. Use [axIDBGetPropDictEntry](#) to get the information about a property dictionary entry. If name is nil, the command returns a list of legal objects that can be used to create property dictionary entries. This is the *objects* attribute of the *o\_propDictEntry* data type. You cannot create a property with the same name as an existing Allegro property.

### **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*. If could not get the entry, it returns nil.

*lt\_validObjects*        List of valid objects to associate with a property

nil                        Could not get the entry.

### **See Also**

[axIDBAddProp](#)

### **Example**

The following example gets the "SIGNAL MODEL" property, and dumps its attributes.

## **Allegro SKILL Reference**

### Database Read Functions

---

```
myprop = axlDBGetPropDictEntry("SIGNAL_MODEL")
myprop->??
  (write nil useCount 0 units nil
    range nil objType "PropDict"
    name "SIGNAL_MODEL"
    dataType "STRING"
    readOnly t
  )
```

## **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 Allegro PCB Editor defined properties only. nil means retrieve both user and Allegro 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

The following example selects the component with refdes "U1," gets its properties using the axlDBGetProperties command, and prints the associated property list it returns. The properties are:

- ❑ ROOM with value D
- ❑ DFA\_DEV\_CLASS with value DIP
- ❑ LEAD\_DIAMETER with value 23 mil.

```
axlClearSelSet()
axlSetFindFilter(?enabled '("noall" "alltypes") ?onButtons "alltypes")
    axlSingleSelectName("component" "U1")
myprops = axlDBGetProperties(car(axlGetSelSet()) '("user" "allegro"))
print myprops
==> ((ROOM "D")
      (DFA_DEV_CLASS "DIP")
      (LEAD_DIAMETER "23 MIL"))
```

## **ax1DBGetDesignUnits**

```
ax1DBGetDesignUnits()  
  => l_value/nil
```

### **Description**

Returns the design units string, accuracy number, and dbuperuu of the active design.

### **Arguments**

None.

### **Value Returned**

<i>l_value</i>	List containing the design units as a string, the accuracy number as an integer, and dbuperuu as an integer
<i>nil</i>	Failed to return the design units, accuracy number, and dbuperuu of the active design.

### **Example**

1. The design Drawing Parameter form shows *User Units* as Millimeter and *Accuracy* as 3.

```
ax1DBGetDesignUnits() ==> ("millimeters" 3 1000)
```

2. The design is setup in Microns to use DBUpperUU of 2000.

```
ax1DBGetDesignUnits() ==> ("microns" 4 2000)
```

## axIDBRefreshId

```
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.



This does NOT update the Allegro database. It updates the cached dbid view of objects in SKILL.

**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

*o\_dbid* SKILL list of *dbids* 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

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.
<i>nil</i>	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*.

## axIDBGetConnect

```
axIDBGetConnect(  
    o_dbid  
    [t_full]  
)  
⇒ l_result/nil
```

### Description

Finds all the elements, including pads and shapes, that are connected to a given *dbid*. Input can be a PIN, VIA, T, CLINE/CARC or CLINE/CARC SEGMENT, and shapes.

If the second argument is *nil* or is not present:

- For pins, vias or Ts, the function returns a list of connected clines.
- For path (clines) or line/arc (segments) returns list of objects connected to either end.
- For shapes same as *t\_full=t*

If the second argument is set to *t*:

- For pins, vias and T, the command returns full connectivity which includes clines, shapes, pins, vias or T's.
- For path (clines) or line/arc (segments) return value is same as *t\_full=nil*
- For shapes list of connected objects which may be clines, shapes, pins, vias or T's.

**Note:** You should set *t\_full* to *t*. The *nil* option operates in its mode due to legacy considerations and is used by Allegro Package Designer applications.

If a segment is passed as an argument, the command does not report inter-path connectivity. Thus only the first and last segment of a path report any connectivity. Internal segments of a path always return nil. This is because the Allegro database connectivity model guarantees that internal segments are always connected to their adjacent segments. The list of segments reported in a path (cline) dbid is how the individual segments are connected.

## Allegro SKILL Reference

### Database Read Functions

---

#### Arguments

*o\_dbid*      A *dbid*, path (cline), line/arc (segment), shape, pin, via or T.  
*t\_full*      t: For full connectivity of pins, vias, or Ts.  
                  nil: Returns connectivity including any connected SHAPES.  
                  Also supports segments.

#### Value Returned

*l\_result*      List of *dbids* connected to *o\_dbid*.  
                  If *o\_dbid* is a CLINE or SEGMENT, then  
                  *l\_result* = (list *list1* *list2*)  
                  where *list1* = nil or elements connected to the first end  
                  *list2* = nil or elements connected to the second end.  
                  For all other objects, returns a list of connections.  
*nil*            Nothing connected to *o\_dbid*.

# axIDBIsBondpad

```
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.

## **axlDBIsBondwire**

```
axlDBIsBondwire(  
    o_dbid  
)  
⇒ t/nil
```

### **Description**

Verifies whether or not the given element is a *bonding wire*.

A bonding wire is a 3D wire object connecting a die pad to either a finger, power ring, or another die pad.

### **Arguments**

*o\_dbid*                    *dbid* of the element to check.

### **Value Returned**

*t*                            *o\_dbid* is a bonding wire.

*nil*                        *o\_dbid* is not a bonding wire.

## **axlDBIsFixed**

```
axlDBIsFixed(  
    o_dbid  
    [g_showMessage]  
)  
⇒ nil or [dbid of 1st element that makes the item fixed]
```

### **Description**

Verifies whether or not the specified database object is fixed. When the FIXED property is present if can either be directly on the object, on a parent (for example, a CLINE is fixed if the NET is fixed) or on a child (for example, a symbol is fixed if its place bounds is fixed).

An object can be fixed by the following:

- Object has the FIXED property or its parent or child objects have the FIXED property.  
For example, group symbol.
- The object (parent or child) has a private database fixed attribute.
- Object is Read-only (typically due to partition enabled).
- Object is a symbol with test points and the FIXED test point flag is set.
- Object is a pin or via with a test point and 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 causes the element to be fixed (could be more then one).

**Note:** Using axlDBCloak with its 'ignoreFixed' option is recommended.

## Arguments

<i>o_dbid</i>	<i>dbid</i> of the element to check.
<i>g_showMessage</i>	Use <code>t</code> to have Allegro PCB Editor display the message if the item is fixed or <code>nil</code> to have no message displayed.

## Value Returned

<i>dbid</i>	<i>dbid</i> of the element causing the object to be fixed.
<i>nil</i>	Object is not fixed.

## Example

```
p = axlSelectByname("SYMBOL" "U1")
ret = axlDBIsFixed(p)
```

## See Also

[axlDBIgnoreFixed](#), [axlDBIsReadOnly](#), [axlDBCloak](#)

## **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.

## **axlDBViaStack**

```
axlDBViaStack(  
    o_dbidVia  
)  
⇒ l_result
```

### **Description**

Returns stacked vias that are located at the location of the provided via. Vias are ordered from top to bottom and includes the provided via.

### **Notes**

Allegro does not store stacked vias but determines it based on a calculation. Future versions of Allegro may store stacked vias in the database or the algorithm may change.

### **Arguments**

<i>o_dbidVia</i>	A via
------------------	-------

### **Value Returned**

<i>nil</i>	No stack or not a via
<i>l_dbidVias</i>	List of vias in the stack

## **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**

<i>o_modinst</i>	AXL <i>dbid</i> of the module instance (the <i>dbid</i> returned by axlDBCreateModuleInstance.)
------------------	---

### **Value Returned**

<i>t_moddef</i>	String containing the name of the module definition.
<i>nil</i>	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 <i>dbid</i> of the module instance (the <i>dbid</i> returned by axlDBCreateModuleInstance.)
-----------	---

### Value Returned

l_loc	List of data describing the location of the module instance. The list syntax is as follows.
-------	---

list(l\_origin, x\_rotation [g\_mirror])

where

- l\_origin: origin of module
- x\_rotation: rotation in degrees \* 1000
- [g\_mirror]: Is a n optional parameter, which is set to t when mirrored

### 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**

<i>o_modinst</i>	AXL <i>dbid</i> of the module instance (the <i>dbid</i> 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**

<i>o_modinst</i>	AXL <i>dbid</i> of the module instance (the <i>dbid</i> returned by axlDBCreateModuleInstance.)
------------------	---

### **Value Returned**

<i>l_nets</i>	List of names of the nets that are treated as exceptions in the module instance.
<i>nil</i>	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. Name of net is an empty string ("").

### **Arguments**

*net\_dbid*                  Net database object.

### **Value Returned**

*t*                  *net\_dbid* is a Dummy Net.

*nil*                  *net\_dbid* is not a Dummy Net.

### **See Also**

[axlIsPinUnused](#)

## **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.

### **See Also**

[axlXSectionGet](#)

### **Examples**

Tests if layer named GND is negative

```
axlIsLayerNegative("GND")
```

## **axlIsPinUnused**

```
axlIsPinUnused(  
    pin_dbid  
)  
⇒ t/nil
```

### **Description**

Determines if the given pin is unused, indicating that it is on a dummy net.

### **Arguments**

*pin\_dbid* Pin database object.

### **Value Returned**

*t* Pin is unused.

*nil* Pin is used.

### **See Also**

[axlIsDummyNet](#)

## axllsitFill

```
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*. Determines if a layer supports voids for shapes.

### **Arguments**

*t\_layer*      Layer name, for example, ETCH/TOP.  
                or  
                just class name ("ETCH")

### **Value Returned**

t	Voids are allowed.
nil	Voids are not allowed.

### **Example**

```
axlOK2Void("ETH/TOP")
```

## **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.
----------------	--

## **ax1DBGetShapes**

```
ax1DBGetShapes(  
    t_layer  
)  
⇒ l_dbid/nil
```

### **Description**

Provides quick access to shapes without access to visibility or find settings.

### **Arguments**

<i>t_layer</i>	Layer name  nil = all layers  <class> = all subclasses of the class  <class>/<subclass> = specified layer
<i>shape_islands_rpt</i>	To output the same list as the Shape Islands Report. Optional.

### **Value Returned**

l_dbid	List of shapes.
nil	Incorrect argument.

### **Examples:**

- Returns all shapes on the design.

```
ax1DBGetShapes(nil)
```

- Returns all shapes on the BOUNDARY layer.

```
ax1DBGetShapes("BOUNDARY")
```

- Returns all shapes on ETCH GND.

```
ax1DBGetShapes("ETCH/GND")
```

- Returns all shapes on ROUTE KEEPOUT.

```
ax1DBGetShapes("ROUTE KEEPOUT")
```

## **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)
```

## **axlShapeArea**

```
axlShapeArea (
    g_mode
    o_shapeId
)
⇒ nil/f_area
```

### **Description**

Area of a shape. Subtracts voids and treats xhatch and unfilled shapes as solid.

**Note:** If using the axl Polygon APIs, the area of the polygons are already part of the polygon structures.

### **Arguments**

<i>g_mode</i>	Must be nil
<i>o_shapeId</i>	Dbid of a shape, rectangle or void

### **Value Returned**

<i>nil</i>	Not a shape
<i>f_area</i>	Area of shape in square design units

## axlStepGet

```
axlStepGet(  
    g_operation  
    g_mapType  
    t_name/o_dbid  
) -> g_return/nil
```

### Description

Obtains STEP mapping data on an object.

### STEP Attributes

Attributes with Modify of "Yes" are recognized by [axlStepSet](#).

NAME	TYPE	Modify	DESCRIPTION
objectType	string	No	"step"
offset_x	double	Yes	Mapping offset in X direction (in design units)
offset_y	double	Yes	Mapping offset in Y direction (in design units)
offset_z	double	Yes	Mapping offset in Z direction (in design units)
rotation_x	double	Yes	Rotation in degrees in X direction
rotation_y	double	Yes	Rotation in degrees in Y direction
rotation_z	double	Yes	Rotation in degrees in z direction
color	int	Yes	This is the override color for the step model.  A value of 0 indicates, the color(s) are taken from the STEP file but not all STEP models have color. Otherwise this is an index into Allegro's color table (see axlColorGet). This value is ignored if the STEP model has color.
step_name	string	No	Step model (relative path)

NAME	TYPE	Modify	DESCRIPTION
type	symbol	No	can be 'primary or 'alt

## Arguments

<i>g_operation</i>	nil - fetch mapping data  'map - return type of mapping (ignores the g_type argument)
<i>g_mapType</i>	should be 'primary or 'alt (a nil is equiv to 'primary)
<i>t_name</i>	may be compdef or a symdef name
<i>o_dbid</i>	a symdef or compdef. For a .dra file this should be nil

## Value Returned

<i>nil</i>	Error or object has no STEP association
<i>if</i> <i>g_operation == 'map</i>	<ul style="list-style-type: none"> <li>■ If object has a STEP association return a list of its current mappings. This list will have 'primary and/or 'alt symbols.</li> </ul>
<i>if g_operation ==</i> <i>nil</i>	<ul style="list-style-type: none"> <li>■ Returns nil if no STEP associated with object and mapType</li> <li>■ Otherwise returns a disembodied property list of mapping attributes (see <a href="#">STEP Attributes</a>)</li> </ul>

## See Also

[axlStepSet](#), [axlStepDelete](#), [axlColorGet](#)

## Examples

- If a .dra design STEP assignment for primary
 

```
result = axlStepGet(nil 'primary nil)
```
- Get primary assignment in a .brd for a symbol definition name
 

```
result = axlStepGet(nil 'primary "CAP10X16MM-RAD")
```
- Report STEP assignments types for same symdef
 

```
result = axlStepGet('map nil "CAP10X16MM-RAD")
```

## Allegro SKILL Reference

### Database Read Functions

---

- Report STEP assignments types for same component definition

```
result = axlStepGet('map nil "CAP")
```

- Report STEP assignments types for same component definition

```
result = axlStepGet('map nil "CAP")
```

- Report STEP assignments types for same component definition using a dbid of a compdef

```
refdes = axlDBFindByName('refdes "C1")  
result = axlStepGet('map nil refdes->compdef)
```

- Report STEP assignments types for same component definition using a dbid of a symdef

```
refdes = axlDBFindByName('refdes "C1")  
; make sure refdes is placed  
when(refdes->symbol  
result = axlStepGet('map nil refdes->symbol->definition))
```

## **axlStepMappedInstance**

```
axlStepMappedInstance(  
    t_name/o_dbid  
) -> t/nil
```

### **Description**

This returns if a symbol or a component instance has a STEP association.

### **Arguments**

<i>t_name</i>	Is a refdes
<i>o_dbid</i>	A symbol or component instance

### **Value Returned**

<i>t</i>	Has a STEP model
<i>nil</i>	Does not have a model (or error)

### **See Also**

[axlStepGet](#)

### **Example**

See if refdes C1 has a STEP model

```
result = axlStepMappedInstance("C1")
```

## axlZoneAccess

```
axlZoneAccess(  
    g_option  
    g_arg  
) -> lt_types
```

### Description

This provides several miscellaneous access functions for zones:

'name	Find zone by name, requires a string for the zone name.  Return is o_zoneDbid.
`point	Given a point as the second argument returns the zone enclosing this point, return is xy.
'trim	Given a zone name or dbid as the second argument trim the zone outline to other zones and the design outline.

### Arguments

g_option	A symbol (see above)
g_arg	This argument is depends upon the option

### Value Returned

t	If successful
nil	If failed
others	Returns depends upon option (see above)

### See Also

[axlZoneCreate](#), [axlGeoPointInShape](#)

### Examples

#### ■ Find zone with name ZONE\_1

```
zone = axlZoneAccess ('name "ZONE_1")
```

## **Allegro SKILL Reference**

### Database Read Functions

---

- Find zone at a point

```
zone = axlZoneAccess('point 3485:-1295.0)
```

- Trim the zone outline (using first example)

```
result = axlZoneAccess('trim zone)
```

---

# Allegro PCB Editor Interface Functions

---

## Overview

This chapter describes the AXL/SKILL functions that give access to the Allegro 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 Allegro 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 15, “Database Create Functions,”](#) and the `axlAddSimpleXXXDynamics` functions described in this chapter.

## Example 1: Dynamic Rubberband

This example loads two circular pads and, 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.

```
axlClearDynamics()

; 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)
;

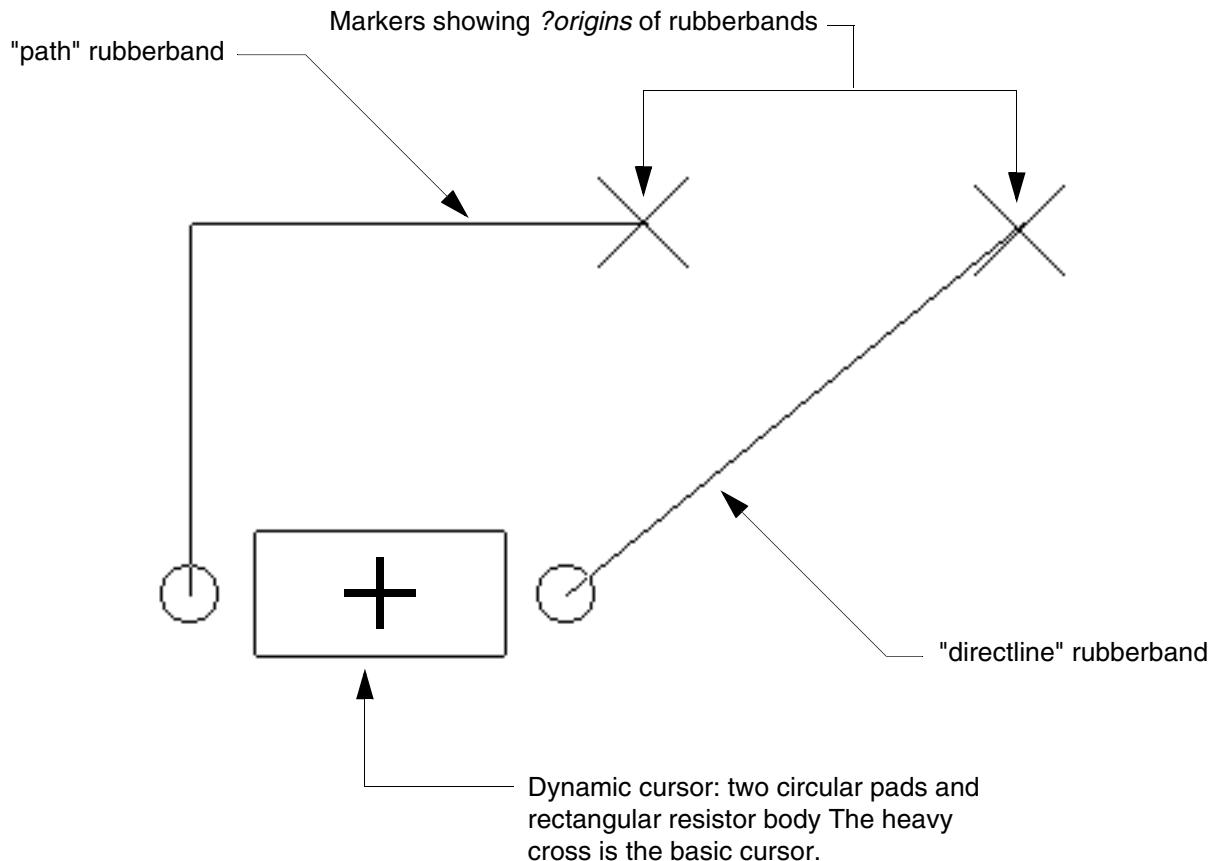
mypoint = axlEnterPoint() ; Ask user for point
```

## Allegro SKILL Reference

### Allegro PCB Editor Interface Functions

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.

The following illustration shows the cursor in a typical position as `ax1EnterPoint` waits for selection of a point.



### Example 2: Dynamic Cursor Rotation

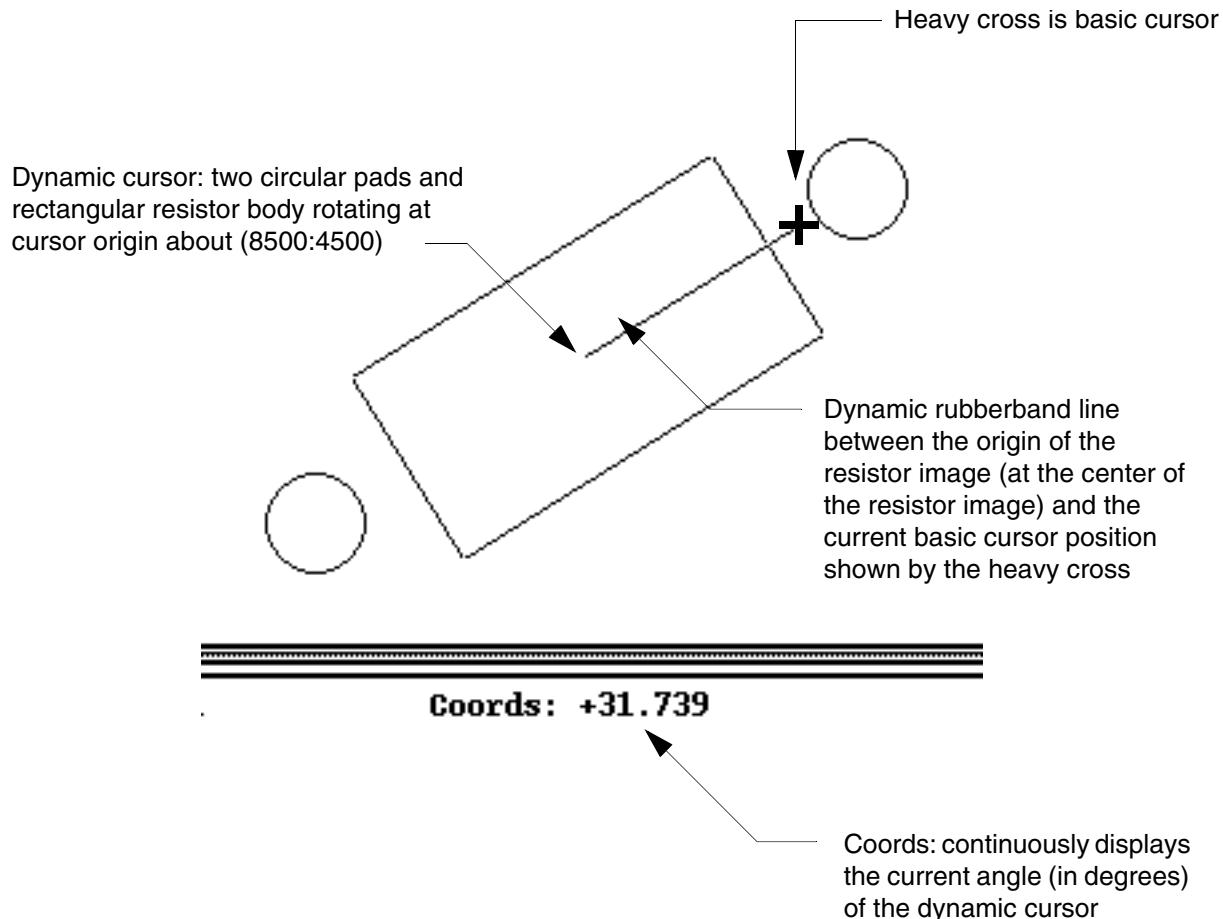
```
axlClearDynamics() ; Clean out any existing cursor data  
  
mypath = axlPathStart(list( -350:0)) ; Start circular pad  
axlPathArcCenter(mypath, 0., -350:0, nil, -300:0)  
  
; Load the first pad into the dynamic cursor buffer  
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)  
  
mypath = 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 SKILL Reference

### Allegro 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.

## Allegro PCB Editor Interface Functions

This section lists Allegro 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 499.

## axlAddSimpleRbandDynamics

```
axlAddSimpleRbandDynamics (
    l_fixed_point
    t_type
    ?origin      l_origin
    ?var_point   l_var_point
    ?lastPath    l_lastPath
    ?width       f_width
    ?color       g_color
)
⇒ t/nil
```

### Description

Loads rubber band dynamics buffer with an element. If dynamics buffer is already loaded, the new element is simply added to the existing buffer. Dynamics buffer is not cleared until [axlClearDynamics](#) is called.

Rubber band dynamics means stretching of elements to the cursor from an anchor point called the `fixed_point`.



***This works in conjunction with the axl<Event> APIs. In particular, no grid snapping, only works when these APIs are called. Do not use this in the axlUIWTimerAdd or with axlTriggerSet callbacks.***

### Arguments

`l_fixed_point`      Fixed point of rubber band. Anchor point from which the dynamic rubberband stretches. The rubberband cursor stretches dynamically from `fixed_point` to current position of the cursor, as moved by the user. The next argument, `t_type`, specifies the shape of the rubberband—part of a path, direct, z-line (a combination of horizontal and vertical), arc, circle, or box.

## Allegro SKILL Reference

### Allegro PCB Editor Interface Functions

---

<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.  directline: add a single line to buffer between <i>fixed_point</i> and <i>var_point</i> .  origin and variable point of <i>var_point</i>  horizline: A single horizontal line.  vertline: A single vertical line  arc: Arc between <i>fixed_point</i> and <i>var_point</i> . Radius varies as cursor moves  "circle": Circle, <i>fixed_point</i> is center and <i>var_point</i> is initial radius. "box": Add a box, <i>fixed point</i> is one corner and the <i>var_point</i> is the opposite corner.  "path": Add two segments whose behavior is controlled by the line lock attributes (axlSetLineLock).  "fixedline": Adds a constant line to cursor buffer, <i>fixed_point</i> and <i>var_point</i> are the two endpoints.
<i>l_origin</i>	Cursor origin. Useful only if you plan on rotating the object, this is the center of its rotation. Also on arcs to control tangency. In most cases this should be nil.
<i>l_var_point</i>	Variable point for rubberbanding.
<i>l_lastPath</i>	Previous path structure. Needed to calculate tangent point if rubberbanding starts at the end of an existing path.
<i>f_width</i>	Optional database width of the rband. Default is 0.0.
<i>g_color</i>	Optional arguments for defining the dynamics' color. Possible choices are: <ul style="list-style-type: none"><li>■ A layer string (i.e. class/subclass) for the layer to be used for deriving the color.</li><li>■ 'ratsnestColor - the color used for ratsnest lines will be used.</li><li>■ 'activeSubclassColor - the color for the active class/subclass is used. If this changes, the color for this rband also changes.</li></ul>

### Value Returned

t	Successfully added data.
nil	No data added.

### Example

A file, `demo_dynamics.il`, in `<cdsroot>/share pcb/examples/skill` demonstrates the various t\_type options.

This example loads two circular pad and, 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:

```
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))

; Loads the resistor body outline in the dynamic cursor buffer
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)
; Ask user to pick angle of rotation about (8500:4500):
axlEnterAngle(8500:4500)
```

See dynamic cursor examples in the section [AXL-SKILL Interface Function Examples](#) on page 499.

### See Also

[axlEnterPoint](#), [axlEnterEvent](#)

## axlAddSimpleMoveDynamics

```
axlAddSimpleMoveDynamics (
    l_origin
    r_path
    t_type
    ?ref_point l_ref_point
    ?color g_color
)
⇒ t/nil
```

### Description

Loads cursor buffer dynamics buffer with an element. If dynamics buffer is already loaded, the new element is simply added to the existing buffer. Dynamics buffer is not cleared until [axlClearDynamics](#) is called.

Cursor buffer dynamics means no stretching of elements. The loaded is attached to the cursor and moves with it.

### Arguments

<i>l_origin</i>	Cursor origin. (see <a href="#">axlAddSimpleRbandDynamics</a> )
<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 lines and arcs are represented as path. Circle is a special case of arc where the start, end points are the same.
<i>l_ref_point</i>	Element rotation reference point.
<i>g_color</i>	Optional argument for defining the dynamics' color. Possible choices are: <ul style="list-style-type: none"><li>■ A layer string (class/subclass) for the layer to be used for deriving the color.</li><li>■ '<code>ratsnestColor</code>' - the color used for ratsnest lines will be used.</li><li>■ '<code>activeLayerColor</code>' - the color for the active class/subclass is used. If this changes, the color for this rband also changes.</li></ul>

### Value Returned

t                    Returned if the data is successfully added.

nil                No data added.

### Example

See dynamic cursor examples, [Example 1: Dynamic Rubberband](#) and [Example 2: Dynamic Cursor Rotation](#), in the section [AXL-SKILL Interface Function Examples](#) on page 499.

## **axlDesignFlip**

```
axlDesignFlip(  
    g_flip  
) t/nil
```

### **Description**

Visually flips the design in the 'y' axis. Maintains current xy view.

**Note:** This command not available if OpenGL is disabled.

### **Arguments**

t	flipped on y axis
nil	unflip

### **Value Returned**

Old flip state. If t flipped (y) if nil normal top view state

### **See Also**

[axlWindowFit](#)

### **Example**

Syntax to implement toggle flipping

```
axlDesignFlip( !axlDesignFlip())
```

## 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's</i> 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 499.

### See Also

[axlGetLastEnterPoint](#), [axlEnterEvent](#)

## 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. Allegro 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-nil, 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.
nil	No angle selected.

### Example

See Example 1 in the section [AXL-SKILL Interface Function Examples](#) on page 499.

## **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 504.

## **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 504.

## **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))
```

## **axlGetLineLock**

```
axlGetLineLock(  
    s_name  
    [g_value]  
)  
==> g_currentValue/ls_names
```

### **Description**

Gets the current settings of the line lock or dynamic control options. Equivalent items is the option control panel for "add" commands. Items currently supported:

- Name: arcEnable  
Value: t/nil  
Description: If t Lock Mode is arc, nil is line.
- Name: lockAngle  
Value: 0, 45, 90  
Description: In degrees where 0 is off (no lock).
- Name: minRadius  
Value: float  
Description: Minimum Radius in user units.
- Name: length45  
Value: float  
Description: Fixed 45 Length value in user units.
- Name: fixed45  
Value: t/nil  
Description: If t Fixed 45 length is enabled.
- Name: lengthRadius  
Value: float  
Description: Fixed radius value in user units.
- Name: fixedRadius  
Value: t/nil  
Description: If t in Fixed Radius mode
- Name: lockTangent  
Value: t/nil  
Description: If t tangent mode is on.

## Arguments

s\_name symbol name of control. nil returns all possible names

## Value Returned

See above.

ls\_names, If name is nil then returns a list of all controls.

## See Also

[axlSetLineLock](#)

## Example

- Return current lock tangent setting

```
axlGetLineLock('lockTangent')
```

- Get all names supported by this interface

```
listOfNames = axlGetLineLock(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 nil, uses default Allegro 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.
nil	Failed to get box data.

## Allegro SKILL Reference

### Allegro PCB Editor Interface Functions

---

#### **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".

#### **See Also**

[axlEnterEvent](#), [axlEnterPoint](#)

## 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.
nil	Failed to get points.

### Example

See the [Enter Function Example](#) on page 504.

## **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**

<i>od_dbid</i>	List of the <i>dbids</i> of figures to be highlighted.
<i>g_permHighlight</i>	Distinguishes temporary highlighting from permanent highlighting using color.  t - use PERM highlight color nil - use TEMP highlight color  The default is nil.

### **Value Returned**

<i>t</i>	Highlighted at least one figure.
<i>nil</i>	Highlighted no figures due to invalid <i>dbids</i> 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`.

## Allegro SKILL Reference

### Allegro PCB Editor Interface Functions

---

- 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 505.

## **axlDehighlightObject**

```
axlDehighlightObject(  
    lo_dbid/g_mode  
    [g_permHighlight]  
)  
⇒ t/nil
```

### **Description**

Use this command to turn off highlighting on an object. Dehighlights the objects whose *dbids* are in *lo\_dbid*. If 'all option is used then *g\_permHighlight* is treated as t (true).

### **Arguments**

<i>lo_dbid</i>	List of <i>dbids</i> of figures to be dehighlighted.
<i>g_mode</i>	'all to dehighlight entire design 'nets dehighlight all nets.
<i>g_permHighlight</i>	Distinguishes temporary highlighting from permanent highlighting using color. t - use PERM highlight color nil - use TEMP highlight color (The default value is nil.)

### **Value Returned**

t	Dehighlighted at least one figure.
nil	Failed to dehighlight any figures.

### **Example**

See [axlHighlightObject](#) on page 524 for examples.

### **See Also**

[axlHighlightObject](#)

## axlMiniStatusLoad

```
axlMiniStatusLoad (
    s_formHandle
    t_formFile / (t_formName t_contents)
    g_formAction
    [g_StringOption]
    [t_restrict]
    [b_reload]
)
⇒ 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, Allegro 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.



**For scripting and performance always use the same `t_formfile` name for an application.**

**Note:** Using these reserved names also causes `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

A description of the in-line [(t\_formName t\_contents)] is contained in the function axlFormCreate.

**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

**b\_reload** If true, form file will not be cached for faster performance. By default, all ministatus options forms are cached as it is expected they will be used repeatedly. A value of true here will prevent this behavior and allow the form to reload from disk.

### 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
```

### See Also

[axlFormCreate](#) on page 819 for further details.

## **axIMiniStatusClose**

```
axIMiniStatusClose(  
    lo_dbid  
) -> t
```

### **Description**

This function closes the active Options panel and returns things to the idle/default options panel for the active application mode.

### **Arguments**

None

### **Value Returned**

*t*

### **Example**

[axIMiniStatusClose](#)

## **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
  [l_ref_point]
)
⇒ t/nil
```

### Description

Adds list of objects to the cursor buffer. These objects are attached to the cursor in xor mode. Origin point establishes cursor position relative to objects in the dynamics buffer.

**Note:** Adding too many objects to the cursor buffer dramatically affects performance.



If you load a symbol definition via `axlLoadSymbol` but does not place the symbol, the definition will, at some time, be deleted from the database.

### Arguments

<code>lo_dbid</code>	List of AXL <code>dbids</code> or single <code>dbid</code> .
<code>l_ref_point</code>	Optional origin point (takes cursor position if not provided).

### Value Returned

<code>t</code>	One or more objects added to the cursor buffer.
<code>nil</code>	No objects added to the cursor buffer.

### Example

- Adds a symbol to the cursor buffer with the symbol origin as a reference point:

```
axlDynamicsObject(symbol_id, symbol_id->xy)
```

- Add a symbol definition to cursor buffer:

```
axlDynamicsObject(symbol_id->defintion, symbol_id->xy)
```

- Load a symbol and add to cursor buffer to the current cursor location:

```
def = axlLoadSymbol("PACKAGE" "dip14")
```

```
when(def axlDynamicsObject(def))
```

## **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**

<i>g_option</i>	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 7-2](#) on page 536 for a list of events with descriptions.

Returns event structure containing the attributes described in [Table 7-1](#) on page 535. Event occurrence controls what attributes are set by all event types, and sets the `objType`, `event` and `time` attributes.

**Table 7-1 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.

**Allegro SKILL Reference**  
Allegro PCB Editor Interface Functions

---

**Table 7-2 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
MOVE	Mouse is moving. Depending upon the amount of time spent in your callback, the system may sum mouse movements to minimize falling behind in processing mouse events.	
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.	xy, xySnap
STOPDRAG	User terminates 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
DONE	User requests the command to complete.	This event cannot be masked.

**Table 7-2 Events**

Event	Description	Attributes/Mask
CANCEL	Respond to this event by terminating your SKILL program (don't call any more <code>axlEnter</code> 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.

### Programming Hints

- 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.
- When *Snap to Object* right mouse button menus are present and the user selects a snap operation:
  - both the `xy` and `xySnap` return the snap result
  - the `snap` argument is ignored
  - normal pick events function normally
- Use `axlSnapDisableAtRMB` if you do not want to snap the menu items.

## 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.
------------------	--

## See Also

[axlEnterPoint](#) and [axlSnapEnableAtRMB](#)

## Example

A complete example is contained in: <cdsroot>/share pcb/examples/skill/axlcore/EnterEvent.il

```
let( (eventMask event, loop)
  eventMask = '( PICK DBLPICK )
  loop = t
  while( loop
    event = axlEnterEvent(eventMask, nil t)
    case(event->type
      ('PICK
       ...
      ('DBLPICK
       ...
      ('DONE
       ; cleanup
       loop = nil)
      )
      )
      )
```

## 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

<i>s_callback</i>	AXL callback function.
none	Unsets the callback function which disables the callback mechanism.

### Value Returned

t	Set SKILL callback function.
nil	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.

## **axISetDynamicsMirror**

sets mirror option for dynamics

```
axlSetDynamicsMirror( g_mirror ) ==> g_olddmirror
```

## Description

Sets the Dynamics mirroring.

## Arguments

`g_mirror`      `g_mirror type`. Possible Values are:

- GEOMETRY: mirror geometry only (same layer)
  - nil: mirror none
  - t: mirror

## Value Returned

old mirror value

#### **See Also**

### **axIAddSimpleMoveDynamics**

## Example

```
axlSetDynamicsMirror(t)
```

## **axlSetDynamicsRotation**

```
axlSetDynamicsRotation(  
    f_angle/nil  
) ==> f_oldangle
```

### **Description**

Sets the Dynamics rotation. If angle is nil then returns current rotation.

### **Arguments**

f_angle	Floating point number
---------	-----------------------

### **Value Returned**

old angle

### **See Also**

[axlAddSimpleMoveDynamics](#)

### **Example**

```
axlSetDynamicsRotation(45.0)
```

## **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**

<i>lo_dbid</i>	List of <i>dbids</i> or a single <i>dbid</i> .
<i>t_file_name</i>	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*):

<i>t_file_name</i>	Name of the temporary file.
<i>x_width</i>	Width, in characters, of the widest text line.
<i>x_line_count</i>	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.

## **axlWindowFit**

```
axlWindowFit(  
    )  
    => l_bBox
```

### **Description**

Zooms in to (or out of) a design fitting it fully on the window. For the Allegro PCB Editor in layout mode, performs a fit on the outline. For the Allegro 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 Allegro PCB Editor command *window fit*.

## **axlZoomBbox**

```
axlZoomBbox (
    x_window
) => bBox
```

### **Description**

`x_window`: window id or nil to currently active window. nil is the active window and 0 is the primary canvas. Allegro currently only supports one additional canvas so the value 1 indicates that canvas.

### **Arguments**

Returns bounding box (`bBox`) of window.

### **Value Returned**

<code>bBox</code>	a list of two xy coordinates indicating upper right and lower left. These are in design units.
-------------------	---

### **See Also**

`axlZoomControl`

### **Examples**

#### ■ BBox of active window

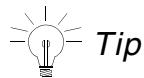
```
axlZoomBbox(nil)
=> ((100 120) (300 320))
```

## axlZoomCenter

```
axlZoomCenter (
    x_window
    xy
)
=> t/nil
```

### Description

Zoom centers on the provided coordinate. It may adjust the point if centering results in the display bounding extents are outside the design extents.



If you wish to zoom and center use axlZoomInOut.

### Arguments

<i>x_window</i>	window id or nil to currently active window (see <a href="#">axlZoomBbox</a> )
<i>xy</i>	Coordinate in design units for centering

### Value Returns

<i>t</i>	if successful
<i>nil</i>	an error

### See Also

[axlZoomControl](#)

### Examples

- Create a secondary window and center it

```
axlZoomControl('create)
axlZoomCenter(1 4000:4000)
```

## **axlZoomControl**

```
axlZoomControl (
  s_option
  [g_arg]
)
--> g_return
```

### **Description**

Manages the multi-canvas feature. Requires OpenGL to be enabled. Id 0 is the main Allegro canvas.

Supported options are:

' create	creates a new canvas. Currently only 1 supported.  Return – If success returns canvas id, if max canvases already exist or multi-window not supported returns nil
' remove	removes secondary canvas. Cannot remove primary canvas (id=0). Requires a canvas id for g_arg.  Return – t if canvas removed, nil if error
' supported	Is multi-window supported.  Return – t if supported, nil not supported
' list	available canvases
' active	returns the active window id. This impacts the Allegro menu Zoom commands and if you pass nil to the axlZoom APIs  Return: integer indicating active window
' swap	swaps the primary and secondary window contents.  Return: t did the swap, nil failed

## Arguments

*s\_option*

see above

*g\_arg*

addition argument some options require, see above

## Value Returned

*g\_return*

depends upon the option, See above

## See Also

[axlZoomBbox](#) [axlZoomPoints](#) [axlZoomCenter](#) [axlZoomWorld](#)

## Examples

- Create secondary canvas

```
axlZoomControl('create')
```

- Remove secondary canvas

```
axlZoomControl('remove 1')
```

## axlZoomFit

```
axlZoomFit (
    x_window
    s_option
) => t/nil
```

### Description

Zoom fits the window. Depending upon the design type fit is define as:

- ❑ logic design (brd, mcm, mdd, etc)
  - Fit to board outline, package and route keepin
- ❑ partition (dps, dpf)
  - Fit to partition boundary
- ❑ symbol (dra) - all visible objects

### Arguments

*x\_window*                    window id or nil to currently active window (see [axlZoomBbox](#))  
*s\_option(visible)*    if logic or partition design fit to visible objects

### Value Returned

*t*                            if successful  
*nil*                        an error

### See Also

[axlZoomControl](#)

### Examples

- Fit primary window to visible objects

```
axlZoomFit(0 visible)'
```

## axlZoomInOut

```
axlZoomInOut (
    x_window
    x_factor
    [xy]
) => t/nil
```

### Description

Zooms window in or out by provided factor around optional coordinate.

### Arguments

<i>x_window</i>	window id or nil to currently active window (see <a href="#">axlZoomBbox</a> )
<i>x_factor</i>	factor to zoom, a positive number zooms in while negative zooms out. 1 is 2x, 2 is 4x, 3 is 8x etc.
<i>xy</i>	optional coordinates to zoom around. If not provided uses center of current window

### Value Returns

<i>t</i>	if successful
<i>nil</i>	an error

### See Also

[axlZoomControl](#)

### Examples

- zoom in by 2x primary window

```
axlZoomInOut(0 1)
```

## **axlZoomPoints**

```
axlZoomPoints (
    x_window
    upperLeft_xy
    lowerRight_xy
) => t/nil
```

### **Description**

Zoom windows by points. The zoom maintains a 1:1 aspect ratio, the coordinates provided will be fitted into the active window size.

### **Arguments**

<i>x_window</i>	window id or nil to currently active window (see axlZoomBbox)
<i>upperLeft_xy</i>	upper left coordinate
<i>lowerRight_xy</i>	lower right coordinate

### **Value Returns**

<i>t</i>	if successful
<i>nil</i>	an error

### **See Also**

[axlZoomControl](#)

### **Examples**

- zoom by points on the primary window

```
axlZoomPoints(0 100:120 4000:4000)
```

## axlZoomToDbid

```
axlZoomToDbid(  
    o_dbid/lo_dbid  
    g_always  
    [x_window]  
)  
⇒ 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. 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.
<i>x_window</i>	Optional window ID or <i>nil</i> to currently active window (see <a href="#">axlZoomBbox</a> ). If no value is provided active window is used.

### 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.

### Example

#### ■ Zoom to U1

```
sym = axlDBFindByName('refdes "U1")  
axlZoomToDbid(sym t)
```

### See Also

[axlZoomManage](#)

**Allegro SKILL Reference**  
Allegro PCB Editor Interface Functions

---

## **axlZoomWorld**

```
axlZoomWorld (   
    x_window  
) => t/nil
```

### **Description**

Zoom world a drawing window

### **Arguments**

*x\_window*                    window id or nil to currently active window (see [axlZoomBbox](#))

### **Value Returns**

<i>t</i>	if successful
<i>nil</i>	an error

### **See Also**

[axlZoomControl](#)

### **Examples**

- world the active window

```
axlZoomWorld (nil)
```

## **axlMakeDynamicsPath**

```
axlMakeDynamicsPath(  
    l_formatedList  
)  
⇒ r_path/nil
```

### Description

This is a convenience function to construct an *r\_path* from a formatted list. axlDBCreate and axlPoly require an *r\_path*.

**Note:** A circle is an arc segment with same end points.



***Passing an illegal format may result in a bad return.***

## Arguments

( *l\_seg1* *l\_seg2* ... ) Each *l\_seg* is:

*g\_clockwise*

```
( l_startPoint l_endPoint [f_width] [l_center]  
[f_radius])
```

*l\_startPoint*: Start point of path.

*l\_endPoint*: End point of path.

*f\_width*: Optional width (default of 0).

*l\_center*: Optional center point if *r\_path* is an arc.

*f\_radius*: Optional radius if *r\_path* is an arc.

If an arc `r_path`, both `l_center` and `f_radius` must be provided.

*g\_clockwise*

### Direction to create arc:

$t \Rightarrow$  create arc clockwise from start to endpoint.

`nil` ⇒ create counterclockwise. Default is counterclockwise.

## Value Returned

r\_path

*dbid* of r\_path.

nil

No r\_path constructed due to incorrect arguments.

## Example

Simple r\_path segment with a width of 20.

```
a = axlMakeDynamicsPath(list(list( 10:10 100:100 20)))
```

**Allegro SKILL Reference**  
Allegro PCB Editor Interface Functions

---

**Allegro SKILL Reference**  
Allegro PCB Editor Interface Functions

---

---

# **Allegro PCB Editor Command Shell Functions**

---

This chapter describes the AXL-SKILL functions that access the Allegro PCB Editor environment and command shell.

## **Command Shell Functions**

This section lists Allegro PCB Editor command shell functions.

## axlGetAlias

```
axlGetAlias(  
    t_alias/nil  
)  
⇒ t_value/1t_names/nil
```

### Description

Requests the value of the specified Allegro PCB Editor alias, *t\_alias*. If given a nil, returns a list of aliases currently set. For compatibility purposes, axlGetAlias returns funkey settings.

### Arguments

*t\_alias*                          Name of the Allegro PCB Editor environment alias.

### Value Returned

<i>t_value</i>	String value of the Allegro PCB Editor environment alias.
<i>1t_names</i>	List of alias names.
nil	Alias not set.

### See Also

[axlSetAlias](#), [axlGetFunckey](#)

### 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.

## axlGetFunckey

```
axlGetFunckey(  
    t_alias/nil  
)  
==> t_value/nil
```

### Description

Requests the value of the specified funckey, *t\_alias*. If given *nil*, returns a list of currently set funckeys.

### Arguments

<i>t_alias</i>	Name of the environment funckey.
<i>nil</i>	Returns list of all current funckeys

### Value Returned

<i>t_value/nil</i>	String value of the environment funckey. Returns <i>nil</i> if the funckey is not set.
<i>lt_names</i>	If passed <i>nil</i> , returns list of funckeys names.

### See Also

[axlSetFunckey](#), [axlGetAlias](#)

### Examples

- Gets the value of the funckey assigned to shifted function key m.

```
alias = axlGetFunckey("m")  
==> "grid"
```

- Return all set aliases.

```
list_alias = axlGetFunckey(nil)  
==> ("-" "+" "m")
```

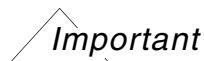
## **axlGetVariable**

```
axlGetVariable(  
    t_variable  
)  
⇒ t_value/nil
```

### **Description**

Requests the value of the specified Allegro 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 Allegro 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**

<i>t_variable</i>	String giving name of the Allegro PCB Editor environment variable.
<i>nil</i>	If <i>nil</i> , returns a list of all set variables in Allegro PCB Editor

### **Value Returned**

<i>t_value</i>	List containing string value of the Allegro PCB Editor environment variable.
<i>nil</i>	Variable not set.
<i>lt_names</i>	List of variable names (returned when <i>nil</i> is passed)

### **See Also**

[axlUnsetVariable](#), [axlSetVariable](#), [axlGetVariableList](#), [axlReadOnlyVariable](#)  
[axlSetVariableFile](#), [axlUnsetVariableFile](#)

## Example

```
menu = axlGetVariable("menuload")
      ==> "geometry"
psmpath = axlGetVariable("psmpath")
      ==> ". symbols"
```

- Gets value of the current menu loaded.  
Variable name is *menuload*.
- 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 Allegro PCB Editor environment variable, *t\_variable*. Unlike `axlGetVariable` this returns a list of strings, if the variable is an array, such as one of Allegro 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 using the SKILL `parseString` command, 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**

<i>t_variable</i>	Name of the Allegro PCB Editor environment variable.
<i>nil</i>	If <i>nil</i> , returns a list of all set variables in Allegro PCB Editor.

### **Value Returned**

<i>t_value</i>	String value of the Allegro PCB Editor environment variable.
<i>nil</i>	Returns <i>nil</i> if the variable is not set.
<i>lt_names</i>	list of variable names, returned when <i>nil</i> is passed as the argument

### **See Also**

[axlGetVariable](#)

## **Allegro SKILL Reference**

### Allegro PCB Editor Command Shell Functions

---

#### **Example**

Gets the value of the Package Symbol search path:

```
path = axlGetVariableList("psmpath")
==> ( "." "symbols" "/cds/root/share pcb/allegrolib/symbols")
```

## **axlJournal**

```
axlJournal(  
    g_option  
)  
==> t_tempFileName
```

### **Description**

This function manages the program's journal file. It has several modes of operation:

g\_option = 'close'

closes current journal file; returns name of closed file

g\_option = <t\_filename>

close current journal file and opens no file, returns t if successful, nil if can't open file.  
Side effect of failure is current journal file is closed.

g\_option = 'name'

returns fullpath name of current journal file, nil if no active journal file



#### **Caution**

***Typically the journal file is buffered. Reading the file while it is open may be unpredictable.***



#### **Caution**

***On Windows, an open file for writing cannot be read. You must close it first.***

### **Argument**

g\_mode                    see above

### **Value Returned**

See above

### **Example**

- Name of file

## **Allegro SKILL Reference**

### Allegro PCB Editor Command Shell Functions

---

```
axlJournal('name')
```

- Open new file in tmp in current directory

```
axlJournal("my_journal")
```

## **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**

<i>t_alias</i>	Name of the Allegro PCB Editor environment alias.
<i>t/nil</i>	<i>t</i> protects the alias, and <i>nil</i> unprotects the alias.

### **Value Returned**

<i>t</i>	Successfully protected or unprotected the alias.
<i>nil</i>	Alias is not set, or the function received invalid data.

### **Example**

```
axlProtectAlias( "F2" t)
```

Protects the F2 function key.

## **axlIsProtectAlias**

```
axlIsProtectAlias(  
    t_alias  
)  
⇒ t/nil
```

### **Description**

Tests if the alias is read-only (or writable). This may also be used with funckeys.

### **Arguments**

*t\_alias*                          Name of the Allegro PCB Editor environment alias.

### **Value Returned**

*t*                                  Alias is protected.

*nil*                                Alias is unprotected or not set.

### **See Also**

[axlIsProtectAlias](#)

## **axlReadOnlyVariable**

```
axlReadOnlyVariable(  
    t_variable  
    [g_Enable]  
)  
==> t/nil
```

### **Description**

This sets, unsets or queries the read-only state of an Allegro PCB Editor environment variable. Once a variable is set read only it cannot be changed. When you set a variable that is unset to read only, the user then cannot set it.

**Note:** Variable names are case insensitive.

### **Arguments**

<i>t_variable</i>	The name of the Allegro PCB Editor environment variable.
<i>g_Enable</i>	<i>t</i> to set read-only; <i>nil</i> to make writable and do not provide if using to test variable read-only state.

### **Value Returned**

<i>t/nil</i>	In query mode ( <i>no g_Enable option</i> ) returns <i>t</i> if variable is read-only and <i>nil</i> if not. If changing the read-only mode, returns <i>t</i> if successful and <i>nil</i> if variable is not currently set.
--------------	--

### **See Also**

[axlGetVariable](#), [axlGetVariableList](#)

### **Examples**

The following example:

- Sets `psmpath` to read-only
- Queries the setting
- Resets `psmpath` to writable

## Allegro SKILL Reference

### Allegro PCB Editor Command Shell Functions

---

#### ■ Query the setting

```
axlReadOnlyVariable("psmpath" t)
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 Allegro 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 Allegro 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

Both `axlSetFunckey` and `axlSetAlias` share the same data storage.

## **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 Allegro PCB Editor, for example, import logic, refresh\_symbol.
  - To set funckeys, see `axlSetFunkey`, `axlGetAlias`, `axlProtectAlias`, `axlIsProtectAlias`, `axlSetAlias` `axlGetAlias`, `axlProtectAlias`, and `axlIsProtectAlias`.

## Arguments

*t\_alias* Name of the Allegro 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

```
ax1SetAlias( "F2" "save")
```

Sets the F2 function key to the save command.

## Example 2

```
axlSetAlias( "~S" nil)
```

**Unsets the Ctrl-S alias.**

## axlSetAlias

```
axlSetAlias(  
    t_alias  
    g_value  
)  
⇒ t/nil
```

### Description

You can set the Allegro 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 Allegro 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

Both `axlSetFunckey` and `axlSetAlias` share the same data storage.

## **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 Allegro PCB Editor, for example, import logic, refresh\_symbol.
  - To set funckeys, see `axlSetFunkey`, `axlGetAlias`, `axlProtectAlias`, `axlIsProtectAlias`, `axlSetAlias` `axlGetAlias`, `axlProtectAlias`, and `axlIsProtectAlias`.

## Arguments

*t\_alias* Name of the Allegro 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

```
ax1SetAlias( "F2" "save")
```

Sets the F2 function key to the save command.

## Example 2

```
axlSetAlias( "~S" nil)
```

**Unsets the Ctrl-S alias.**

## **axlSetFunckey**

```
axlSetFunckey(  
    _alias  
    g_value  
)  
==> t/nil
```

### **Description**

Works similar to `axlSetAlias` except allows alpha-number keys to work like function keys (no Enter key required). See `axlSetAlias` for complete documentation.

- Funckey settings only apply to current session. They are not saved to user's local `env` file.
- Funckey changes do not affect programs launched from Allegro PCB Editor: for example, `import logic` or `refresh_symbol`.

### **Arguments**

<code>t_alias</code>	name of the Allegro environment alias.
<code>g_value</code>	Value to which the environment alias is to be set. Can be a string, or nil.

### **Value Returned**

<code>t</code>	Returns <code>t</code> if successful.
<code>nil</code>	Returns <code>nil</code> if invalid data type of alias is marked read only.

### **See Also**

[axlGetFunckey](#), [axlIsProtectAlias](#), [axlIsProtectAlias](#), [axlSetAlias](#)

### **Examples**

Set the funckey alias to move

```
axlSetFunckey( "m" "move" t)
```

Unset the move

## **Allegro SKILL Reference**

### Allegro PCB Editor Command Shell Functions

---

```
axlSetFunckey( "m" nil)
```

## **axlSetVariable**

```
axlSetVariable(  
    t_variable  
    g_value  
)  
⇒ t/nil
```

### **Description**

Sets the Allegro PCB Editor environment variable with name given by the string *t\_variable* to the value *g\_value*. The *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 Allegro PCB Editor.

**Note:** 511 is the maximum list long (*l t\_variable*).

### **Notes:**

- Variable names and values can change from release to release.
- Variable settings only apply to current session. They are not saved to local env file for the user.
- Variable changes do not effect programs launched from Allegro PCB Editor. For example, import logic, refresh\_symbol
- Many Allegro operations are done via batch operations such as items in File Import/Export, artwork, axlRunBatchDBProgram, and so on. These operations do NOT see variables changed (like PSMPATH) by this call or by the Allegro set command.

## Arguments

<i>t_variable</i>	String giving the name of the Allegro PCB Editor environment variable.
<i>g_value</i>	Value to which the environment variable is to be set. Can be a string, int, t, or nil.

## Value Returned

<i>t</i>	If successful
<i>nil</i>	If invalid data type of variable is marked read-only

## See Also

[axlGetVariable](#), [axlReadOnlyVariable](#), [axlSetVariableFile](#), [axlUnsetVariable](#),  
[axlUnsetVariableFile](#)

## Example

Sets new library search path libpath.

```
(axlSetVariable "libpath" "/mytools/library")
⇒ t
libraryPath = axlGetVariable ("libpath")
⇒ "/mytools/library"
```

Using list mode:

```
axlSetVariable("psmpath" '("." "symbols"))
    ==> t
    axlGetVariableList("psmpath")
    ==> ("." "symbols")
```

## **axlSetVariableFile**

```
axlSetVariableFile(  
    t_variable  
    g_value  
)  
==> t/nil
```

### **Description**

Sets and saves to file Allegro environment variable. This operates the same as [axlSetVariable](#) except it also saves the setting to the user's local environment file.

Variable is added in the preference section of the env file.



***On Windows, updating the environment file on disk can cause performance issues if this interface is used heavily.***

### **Arguments**

<i>t_variable</i>	Name of the Allegro environment variable.
<i>g_value</i>	Value to which the environment variable is to be set. Can be a string, int, t, or nil.

### **Values Returned**

<i>t/nil</i>	Returns <i>t</i> if successful. Returns <i>nil</i> if invalid data type of variable is marked read-only.
--------------	--

### **See Also**

[axlSetVariable](#), [axlUnsetVariableFile](#)

# axIShell

```
axlShell(  
    t_command  
)  
⇒ t
```

## Description

Issues the Allegro 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 Allegro PCB Editor releases.

## Arguments

*t\_command* Allegro PCB Editor shell command or commands.

## **Value Returned**

#### **See Also**

## ax1ShellPost

## Example 1

```
(axlShell "status")  
⇒ t
```

Displays Allegro PCB Editor Status form from AXL-SKILL.

## Example 2

```
axlShell("zoom points; pick 0 0; pick 100 100")
```

**Chained command example:**

# axIShellPost

```
axlShellPost( t_command ) ==> t
```

## Description

This works similar to `ax1Shell` except it first requires a return from the SKILL interpreter before executing the command(s). It should only be used in the special circumstance where you want to do some processing in SKILL, execute an Allegro PCB Editor interactive command and have that command be left active for the user. If more than one command is embedded in post command then subsequent commands should be prefixed with an underscore to inhibit scripting. For example:

```
axlShellPost("zoom points; pick 10 20")
```



***Do not attempt to use this as a method to override an existing Allegro PCB Editor command with SKILL code and then call the original command. An infinite loop will result.***



*This function may not be portable across Allegro PCB Editor releases.*

## Arguments

*t\_command* Allegro PCB Editor shell command or commands.

## Value Returned

#### **See Also**

ax1Shell

## EXAMPLES

Over the move command to print hello and then let user move objects.

```
axlCmdRegister( "mymove" 'testSkill ?cmdType "interactive")
procedure( testSkill()
    printf("Hello mymove\n")
    axlShellPost("echo hello from post; _move")
    printf("Hello after-mymove\n")

)
Output -- showing deferred execute:
Hello mymove
Hello aftermove
hello from post
Select element(s) to move.
```

## axlUnsetVariable

```
axlUnsetVariable(  
    t_variable  
)  
⇒ t
```

### Description

Unsets the Allegro 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 Allegro PCB Editor environment variable.

### Value Returned

t      Always returns t.

### See Also

[axlSetVariable](#)

### Example

```
(axlUnsetVariable "libpath")  
⇒ "/mytools/library"  
  
libraryPath = (axlGetVariable "libpath")  
⇒ nil
```

Clears the library path libpath when its current value is /mytools/library.

## **axlUnsetVariableFile**

```
axlUnsetVariableFile(  
    t_variable  
)  
==> t
```

### **Description**

Unsets the value of specified Allegro environment variable. Works the same as [axlUnsetVariable](#) plus it also updates the local environment file of the user with the change.



***On Windows, updating the environment file on disk can cause performance issues if this interface is used heavily.***

### **Arguments**

*t\_variable*                    String giving the name of the Allegro environment variable.

### **Value Returned**

*t*                            Always returns t.

### **See Also**

[axlSetVariableFile](#), [axlSetVariable](#)

**Allegro SKILL Reference**  
Allegro PCB Editor Command Shell Functions

---

---

## **IC Packaging Commands**

---

The functions listed in this chapter are available only in Allegro® Package Designer+ (APD+).

## **axlChangeLayer**

```
axlChangeLayer(  
    lo_dbid/o_dbid  
    t_newLayer  
    [o_padStackDbid]/[t_padstackname]  
)  
==> t/nil
```

### **Description**

Changes layer for lines, clines or segments, shapes, and text. Functionality offered, at the global level, matches the Allegro PCB Editor `change` command, but can differ in certain areas.

If moving clines or cline segments across layers, you should provide a via to be used to maintain connections. Via must meet constraint rules. For via to be accepted it must have:

- pads on start and destination layers
- be in the constraint via list for the net and location

If the provided padstack is not acceptable, system will select an acceptable padstack based upon the via list for net and location.

**Note:** If you need to change the layer of multiple etch objects, it is more efficient to pass them as a list of dbids then to call this function for each dbid.

### **Arguments**

<i>lo_dbid/o_dbid</i>	a single dbid or list of dbids
<i>t_newLayer</i>	new layer for placing dbid
<i>o_padStackDbid</i>	if moving clines across layers, allegro will add a via to maintain connection. This is that via. If this argument is not provided the system default via is used.
<i>t_padStackName</i>	name of padstack to be used

### **Value Returned**

*t* if succeeded, *nil* if failure

### ***Failures***

For debug purposes set axlDebug(t) to see additional messages.

- dbid is of a unsupported type
- illegal option types
- target layer matches object current layer

### **Examples**

#### **■ Move an object to ETCH/BOTTOM**

```
; ashOne is a selection utility found at <cdsroot>/pcb/examples/skill/ash-fxf/  
ashone.il  
dbid = ashOne()  
; pick an object (set find filter)  
result = axlChangeLayer(dbid "ETCH/BOTTOM")  
; or if moving clines  
result = axlChangeLayer(dbid "ETCH/BOTTOM" "PAD60SQ36D")
```

### **See Also**

[axlTransformObject](#), [axlDBChangeText](#), [axlChangeWidth](#)

## **axlCNSAssemblyModeGet**

```
axlCNSAssemblyModeGet(  
    nil  
) => ls_constraints  
  
axlCNSAssemblyModeGet(  
    'all  
) => lls_constraintNModes  
  
axlCNSAssemblyModeGet(  
    s_name/t_name  
) => s_mode/nil  
  
axlCNSAssemblyModeGet(  
    s_name/t_name  
    'print  
) => t_name/nil
```

### **Description**

This retrieves the current assembly DRC mode(s). Modes determine if a particular constraint is on or off. These DRC modes apply to the entire design. Use axlCNSAssemblyModeGet(nil) to determine the set currently supported assembly modes. The 'print mode offers the name shown in reports like show element.



***Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant from release to release.***

## Arguments

<i>nil</i>	returns all modes that are in the spacing domain
' <i>all</i>	returns all checks and current mode
<i>s_name</i>	symbol name of check.
<i>t_name</i>	string name of check
' <i>print</i>	printable constraint name option

## 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</i> <i>s_mode</i> ) ...)
<i>s_mode</i>	mode 'on, or 'off
<i>t_name</i>	the printable constraint name

## Examples

- Get current list of assembly constraints  
`axlCNSAssemblyModeGet(nil)`
- Get list of settings for all assembly constraints  
`axlCNSAssemblyModeGet('all)`
- Get current mode of Die to Finger Spacing check  
`axlCNSAssemblyModeGet('die_to_finger)`

## See Also

[axlCNSAssemblyModeSet](#), [axlCNSGetAssembly](#)

## **axlCNSAssemblyModeSet**

```
axlCNSAssemblyModeSet(
```

```
    t_name/s_name  
    t_mode/s_mode  
)  
=> t/nil
```

```
axlCNSAssemblyModeSet(
```

```
    'all  
    t_mode/smode  
)  
=> t/nil
```

```
axlCNSAssemblyModeSet(
```

```
    l_constraintNModes  
    t_mode/smode  
)  
=> t/nil
```

```
axlCNSAssemblyModeSet(
```

```
    ll_constraintNModes  
)  
=> t/nil
```

### **Description**

This command sets the current DRC modes (on/off) for checks in the area of assembly constraints. These modes are global.

To determine the constraints modes currently supported, use `axlCNSAssemblyModeGet(nil)`.

Several interfaces are supported for this command, all checks ('all), individual checks (`t_name`), list of checks within a mode '`(s_name ...)`' `t_mode/s_mode`', (`t_name ...)` `t_mode/s_mode`, and sets of checks via a list of: '`((s_name/t_name s_mode/t_mode) ...)`' .

The constraints names may be passed as a symbol or a string. For performance reasons, either do all your updates in a single call or wrap individual changes in the map API (see [axlCNSMapUpdate](#)).



**Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant 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 or 'off.
<i>t_mode</i>	string mode setting "on" or "off".
'all	set all checks for assembly design rule checks (including online wire rules)

## Value Returned

<i>t</i>	if succeeds
<i>nil</i>	if failed

## Examples

- Turn all constraints off

```
axlCNSAssemblyModeSet('all 'off)
```

- Turn Die to Finger Spacing check on

```
axlCNSAssemblyModeSet('die_to_finger 'on)
```

- Turn two constraint to on

```
axlCNSAssemblyModeSet('((via_to_package_edge die_to_finger) 'on)
```

- Set various constraints to different modes

```
axlCNSAssemblyModeSet( '((die_to_finger off) (via_to_package_edge 'on)) )
```

## See Also

[axlCNSAssemblyModeGet](#), [axlCNSGetAssembly](#), [axlCNSMapUpdate](#)

## **axlCNSGetAssembly**

```
axlCNSGetAssembly(
  t_cset
  t_layer
  s_constraint
  [g_string]
)
=> g_value/nil

axlCNSGetAssembly(
  t_cset
  t_layer
  nil
  [g_string]
)
=> ll_nameValue/nil

axlCNSGetAssembly(
  nil
  nil
  nil
)
=> ls_cnsTypes
```

### **Description**

Obtains an Assembly cset values. In the first operational mode, the command obtains the value of an assembly constraint given a cset and a layer. In second mode of operation, it obtains all assembly constraint as name-value pairs for a cset on a layer. For the last mode, a list of all supported assembly constraints is obtained by passing three `nil` values to the interface:`axlCNSGetAssembly(nil nil nil)`

**DATA TYPES** — Unless otherwise specified constraints are in current design units.

## Arguments

<i>t_cset</i>	Name of a assembly cset. Use "" for design-level constraints.
<i>t_layer</i>	ETCH layer name (e.g "ETCH/TOP" or "TOP"). If <i>nil</i> is specified, gets constraints on all layers.
<i>s_constraint</i>	Name of constraint. If <i>nil</i> , returns a set of symbol/value pairs of all constraints.
<i>g_string</i>	By default, returns value in the native units of the constraint. If <i>g_string</i> is set to <i>t</i> , the data is returned as a string.

## Value Returned

<i>g_value</i>	Value of constraint in design units
<i>ll_nameValue</i>	Maple value pairs of assembly constraint symbol and constraint value for all assembly ( <i>s_constraint g_value</i> ). '((wire_len_min 10.0) (wire_len_max 50.0) ...)
<i>ls_cnsTypes</i>	List of supported assembly constraint names
<i>nil</i>	Indicates an error

## Examples

- Get min wire length in design cset  

```
axlCNSGetAssembly("") nil 'wire_len_min)
```
- Get all assembly constraints for "V" cset, bottom layer  

```
axlCNSGetAssembly("V" "BOTTOM" nil)
```
- Get min shape constraint list for "V" cset, top layer  

```
axlCNSGetAssembly("V" "TOP" 'min_shape_check)
```
- Get supported Assembly constraint symbols  

```
axlCNSGetAssembly(nil nil nil)
```
- Fetch all layers and constraints of Assembly cset V  

```
cset = "V"      ;; D
foreach(subclass axlSubclassRoute()
layer = axlCNSGetAssembly(cset subclass nil)
printf("\nLAYER=%s\n\tconstraints=%L\n" subclass, layer)
)
```

## **Allegro SKILL Reference**

### IC Packaging Commands

---

#### **See Also**

[axICNSSetAssembly](#)

## **axlCNSSetAssembly**

```
axlCNSSetAssembly(
    t_object/nil
    t_layer/nil
    s_constraint
    g_value
    [s_object_type]
)
==> t/nil
```

```
axlCNSSetAssembly(
    t_object/nil
    t_layer/nil
    ll_constraintValues
    nil
    [s_object_type]
)
==> t/nil
```

### **Description**

Updates assembly cset, wire profile or symbol constraint values. By passing `nil` at the appropriate argument, values for all csets and all layers may be changed.

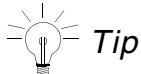
By default, object type is a cset. To update values on wire profile, use wire profile as an object name and pass fifth argument as '`PROFILE`'. To update values on symbols (dies, spacers or interposers) use refdes as an object name, and specify the fifth argument as '`SYMBOL`'.

DATA TYPES — See [axlCNSGetAssembly](#) for the data type of each constraint.

Design Units:

- A number (integer or floating point) where units is current design units. Must not exceed accuracy of the design.
- Unit less string where accuracy cannot exceed db accuracy.
- String with units, data converted to current design units.

Boolean: Use `t/nil` or "true"/"false".



For list of Assembly constraints see [axlCNSGetAssembly](#).



***This does NOT change override values. For example, you can set wire\_len\_min value in all csets but if the user has applied it to a wire profile as an override, it will still be used for those items.***

## Arguments

<i>t_object</i>	cset name, use " " for design level constraint. Use <code>nil</code> to apply change to all cset.
	wire profile name
	die refdes
<i>t_layer</i>	ETCH layer name (e.g "ETCH/TOP" or "TOP"). If <code>nil</code> apply change to all layers - applicable to by layer constraints only. Use <code>nil</code> for constraints that cannot be set by layer.
<i>s_constraint</i>	Constraint symbol to change. Use <code>axlCNSGetAssembly(nil nil nil)</code> for list of permissible values.
<i>g_value</i>	Value to update. For data type see above for "DATA TYPES".
<i>ll_constraintValues</i>	Multiple values may be updated by passing a list of lists for the third argument.  <code>'((s_constraint g_value) ... )</code>
<i>s_object_type</i>	By default, object type is cset. Use ' <code>SYMBOL</code> ' for die, ' <code>PROFILE</code> ' for wire profile

## Value Returned

- `t` if succeeds
- `nil`, in case of an error. This value is returned if:
  - cset name does not exist
  - layer does not exist
  - constraint does not exist
  - illegal value for constraint

## Examples

- Set min shape check on all constraint sets and layers  
`axlCNSSetAssembly(nil nil 'min_shape_check 50)`
- Set min void check on all layers on design level  
`axlCNSSetAssembly("") nil 'min_void_check 60)`
- Set min wire length on wire profile PROFILE1  
`axlCNSSetAssembly("PROFILE1" nil 'wire_len_min 75 'profile)`
- Set die overhang x on die U1  
`axlCNSSetAssembly("U1" nil 'die_overhang_x 50 'symbol)`

## See Also

[axlCNSGetAssembly](#)

## **axlCreateDeviceFileTemplate**

```
axlCreateDeviceFileTemplate (
    t_deviceName
    t_CLASS
    l_pinList
) -> t/nil
```

### **Description**

This creates a template device file providing same functionality as the create device command in the symbol editor. Normally you would use [axIDBCreateComponent](#) to create a device file if in the board.

### **Arguments**

<i>t_deviceName</i>	Name of device file (no file extension or path)
<i>t_CLASS</i>	Class of device (for example, IC, IO, etc.)
<i>l_pinList</i>	List of pins. Can be any combination of either pin dbid or pin numbers. If a pin dbid will filter out mechanical pins

### **Value Returned**

*t* - file created

*nil* - an issue

### **See Also**

[axIDBCreateComponent](#)

## **axlCompAddPin**

```
axlCompAddPin(  
    o_comp  
    g_absLoc  
    o_pin/lo_pins  
) => t/nil
```

### **Description**

This function adds one or more pins to the specified component. Pins are added to the corresponding component and symbol definition, with changes being reflected in all instances of those definitions.

### **Arguments**

<i>o_comp</i>	Dbid of component instance to which pins should be added.
<i>g_absLoc</i>	If true, locations and rotations for pins are absolute values in the design space. If <i>nil</i> , these values are relative to the origin of the unmirrored, unrotated symbol definition.
<i>o_pin/lo_pins</i>	Structure, or list of structures, defining the pins to be added. These objects are defstructs as defined below.

Defstruct used to define a pin. Use `make_axlCompPinRecord`.

Required Elements are:

- *s\_pinUse* – Pin use code for this pin. Must be one of the following symbols:

- UNSPEC
- POWER
- GROUND
- NC
- LOADIN
- LOADOUT
- BI
- TRI

- OCA
- OCL.

- *n\_swapCode* – Swap group code for this pin. A swap code of 0 means this pin is not swappable. Otherwise, all pins with the same swap code are swappable. This value is not used for co-design components, as all co-design comp pins are considered swappable.
- *l\_location* – X/Y coordinate location for the pin. Absolute or relative to the symbol def origin, as specified by *g\_absLoc*.
- *n\_rotation* – Angular rotation of this pin. Absolute or relative to the symbol definition origin, as specified by *g\_absLoc*.

Optional Elements are:

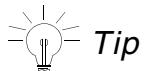
- *pinNumber* – Physical pin number to assign to this pin. Must be unique across all pins of the component. If no pin number is provided, the pin number will be computed based on the pin numbering scheme for the component. If the numbering scheme is 'Customized', as is the default for co-design objects, then the tool will assign the first unused integer as the pin's number (1, 2...).
- *pinName* – Logical pin name for the pin. Not used for power/ground pins. If not provided, the pin name for signal pins will be the same as the physical pin number.
- *verilogPort* – Verilog port name for the pin. Not used for power/ground pins. If not provided, there is no verilog port name for the pin and the function pin name is used.
- *net* – Net name or dbid to assign this pin to when created. If nil, pin will be created on a dummy net and can be assigned later.
- *padstack* – Padstack name or dbid to use for this pin. If no padstack is supplied, the padstack already in use for pins of this component will be used for this pin as well.

Co-design Elements are:

- *codesignNet* – Net name for this pin in the secondary design space. For example:
  - For a co-design die in a package, this is the pin's net in the IC design.
  - For a co-design package in a board, this is the pin's net in the package.
- *codesignPad* – Pad/Cell name for this pin in the secondary design space. For example:
  - For a co-design die in a package, this is the pin's LEF bump macro.
  - For a co-design package in a board, this is the pin's padstack in the package.

### Value Returned

- `t` if the pin(s) were added.
- `nil` if there was an error adding any of the pins, for example, if a pin would be placed outside the extents of the symbol or drawing.



#### *Tip*

If many pins are to be added, it is more efficient to pass the entire list to this function to process them in one call than to call `axlCompAddPin` with each individual pin.

### See Also

[axlCompDeletePin](#), [axlCompMovePin](#)

## **axlCompDeletePin**

`axlCompDeletePin(o_pin/lo_pins) => t/nil`

### **Description**

This function deletes the specified pin(s) from the parent component and symbol definition. As a result, the pin will also be deleted from all instances of these definitions and not just the instance the pin passed in belongs to.

### **Arguments**

*o\_pin/lo\_pins*      SKILL dbid of the pin to be deleted, or a list of pins to be deleted.

### **Value Returned**

- `t` if the pin(s) were deleted.
- `nil` if there was an error deleting any of the pins, for example, if a pin had the fixed property.

**Note:** If many pins are to be delete, it is more efficient to pass the entire list to this function to process them in one call than to call [axlCompDeletePin](#) with each individual pin.

### **See Also**

[axlCompAddPin](#), [axlCompMovePin](#)

## **axlCompMovePin**

```
axlCompMovePin(  
    o_pin/lo_pins  
    ?move          l_deltaPoint  
    ?groupMirror   t/nil  
    ?groupRotation f_angle  
    ?rotOrigin     l_rotatePoint  
    ?pinRotation   f_deltaAngle  
) => t/nil
```

### **Description**

This function moves the specified pin(s) by the specified delta x/y, rotation, and mirror. Rotation and mirror, if provided, are applied around the `rotatePoint` (defaults to 0,0 if not provided). A rotation to apply to each individual pin may also be provided.

### **Arguments**

<i>o_pin/lo_pins</i>	SKILL dbid of the pin to be moved, or a list of pins to be moved.
<i>move</i>	X/Y delta to move each pin in the set by.
<i>groupMirror</i>	t if the position of each pin in the group should be mirrored around the supplied <i>rotOrigin</i> .
<i>groupRotation</i>	angle of rotation that should be applied to each pin in the group to determine its new final location. Applied around <i>rotOrigin</i> .
<i>rotOrigin</i>	X/Y origin for group mirror and rotation application.
<i>pinRotation</i>	Rotation to apply to individual pins once placed at new location.  For example, if you are moving a pin from the north side to west side, and the pin is rectangular, you may wish to specify a rotation of 90 degrees to keep the same orientation of the pin rectangle relative to the nearest die edge.

### **Value Returned**

t if the pin(s) were moved.

nil if there was an error moving any of the pins, for example, if a pin had the fixed property or would be placed outside the extents.

### **Notes:**

- If many pins are to be moved, it is more efficient to pass the entire list to this function to process them in one call than to call `axlCompMovePin` with each individual pin.
- When applying multiple transforms at once to the set of pins, the operations are performed in the following order:
  - a. group mirror is performed for the group about the specified `rotOrigin`.
  - b. group rotation is then applied, also around the `rotOrigin`.
  - c. move delta is then applied.

This is important to keep in mind so that you achieve the desired end position for all pins being moved, and so that you use the correct `rotOrigin` point.

## See Also

[axlCompAddPin](#), [axlCompDeletePin](#)

## **axlComponentChangeClass**

```
axlComponentChangeClass(  
    s_devType/o_compDef  
    s_class  
) -> t_oldClass/nil
```

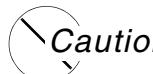
### **Description**

This command changes the component class of a component definition.

Do not change class for the component definitions with component class as MECHANICAL, PLATING\_BAR or DRIVER\_CELL. Also you may not change an object to those classes.

To get the dbid of a component definition,

- For symbol instance, cd = syminst->component->compdef
  - Note:** A symbol may not have a component instance (for example, mechanical).
- For component instance, cd = component->compdef



### **Caution**

***Running a Cadence or 3rd party ECO restores the original component class.***

## Arguments

<i>s_devType</i>	Name of the device.
<i>o_compDef</i>	dbid of a component definition.
<i>s_class</i>	Name of the new component class. Possible values are IC, IO, or DISCRETE

## Value Returned

<i>t_oldClass</i>	Old component class is returned in case of success.
nil	Returned in case of an error.

## Example

```
axlComponentChangeClass("DIP14" "DISCRETE")
```

## See Also

[axlDBCreateComponent](#)

## **axlCompSetPinAttributes**

```
axlCompSetPinAttributes(
  o_pin/lo_pins
  ?number      t_pinNumber
  ?name        t_pinName
  ?use         t_pinUse
  ?padstack   t_padstack/g_padstack
  ?rotation    f_rotation
  ?swapCode   n_swapCode
) => t/nil
```

### **Description**

This function modifies attributes of the specified pin(s) at the symbol and component definition level. Things like the padstack, pin number, and swap code (see below for full list of supported attributes). All update items, if not set, will be left at the pin's existing value.

## Arguments

<i>o_pin/lo_pins</i>	SKILL dbid of the pin to be modified, or a list of pins to be modified.
<i>number</i>	New pin number for pin.  It is recommended that this parameter must be used only with single pins sent in, as pin numbers MUST be unique.
<i>name</i>	Function pin name for the pin.  Since the actual database function, <code>pinname</code> , is a key for the function pin definition, this must be unique within the function definition. However, if a non-unique value is supplied, or the string is not valid for a function pin name (such as it contains lower case characters), the name supplied here is stored as the <code>Verilog_Port_Name</code> for the pin and a legal unique <code>PinName</code> is derived from this name to generate a unique key for the database.
<i>use</i>	New pin use for the pin.  If changing the pin use from power/ground to a signal pin, function pins are created. In the reverse scenario — changing <code>pinuse</code> from signal to power/ground — function pins are removed.
<i>padstack</i>	New padstack to be used for the specified pin.
<i>rotation</i>	New rotation of pin (on the specific symbol instance as currently placed in the design). Symbol definition pin is updated to compensate for final instance rotation.
<i>swapCode</i>	New swap code group index for pin.

## Value Returned

<code>t</code>	Returned when at least one pin is modified.
<code>nil</code>	Returned in case there is an error modifying any of the pins.

## See Also

[axlCompAddPin](#), [axlCompDeletePin](#), [axlCompMovePin](#)

## **axlDBIsBondingWireLayer – Obsolete**

This is an obsolete function. Bonding wire layers have been replaced by die stack layers. Use axlDBIsDieStackLayer to check whether a layer is a die stack layer.

## **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.

## **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.

## axlGetDieType

```
axlGetDieType(  
    o_componentDBID  
)  
==> t_dieType
```

### Description

Returns the die attachment type for a given die component in a Cadence packaging tool (APD+). A die is considered to be an IC class component in the database. Currently, the supported attachment types include the following:

- WIREBOND
- FLIP CHIP

### Arguments

*Component DBID*      *dbid* handle of the die component to query.

### Value Returned

<i>t_dieType</i>	If successful.
<i>nil</i>	If failed (non-die object passed or not a packaging product).

### Examples

```
axlGetDieType (myComp)  
==> "FLIP CHIP"
```

## axlGetMetalUsageForLayer

```
axlGetMetalUsageForLayer(  
    l_layers  
    [l_extents]  
    [g_positive]  
    [exclude_pins_vias]  
)  
==> resultStruct/nil
```

### Description

Computes the percentage metal coverage on the layers specified in *l\_layer(s)* (combination of all layers listed) in the area specified in *l\_extents*. If no extents are provided, then the geometry outline extents of the design are used.

Negative layer metal coverage can be computed by passing nil for *g\_positive*, if processing negative artwork layers such as solder mask layers.

### Arguments

<i>l_layers</i>	A single layer or list of layers to compute the (combined) metal usage on. For example, passing list("CONDUCTOR/TOP" "PIN/TOP") will compute the coverage of all conductor + pin objects on the top layer. It will not compute the two individually.
<i>l_extents</i>	BBox region to compute metal coverage in. If not supplied, the tool will compute based on the substrate geometry outline's extents or, if no outline is present, the database extents.
<i>g_positive</i>	Whether the layer(s) being processed are positive or negative layers. Defaults to true.
<i>exclude_pins_vias</i>	Excludes pins and vias for ETCH class; returns copper area excluding pins and vias. Default is nil.
	For example,

### Value Returned

- *resultStruct/nil* – *resultStruct* is a defstruct containing for elements:
- *areaUnits* – Units in which the area was computed and returned.
- *regionArea* – The area of the extents region the tool used (user supplied or drawing extents).

## **Allegro SKILL Reference**

### IC Packaging Commands

---

- `metalArea` – The total metal area in the region checked on the layers indicated.
- `percentMetalCoverage` – The percentage of the extents region covered by metal.  
Calculated as  $((\text{metalArea} / \text{regionArea}) * 100.0)$
- `nil` is returned if there is an error, with error printed to command line.

## **axlGetWireProfileDefinition**

list axlGetWireProfileDefinition(profileName)

### **Description**

Given a bonding wire profile name, this will returns its definition information.

### **Arguments**

*profileName*      Name of the profile definition being queried. If this argument is nil, definitions of all profiles in design are retrieved.

### **Value Returned**

Structure of information describing the profile definition (material, 3d points list, etc).

In case of failure, an error string is returned.

## **axlAddAutoAssignNetAlgorithm**

```
axlAddAutoAssignNetAlgorithm(t_algorithm t_displayName)
==> t/nil
```

### **Description**

This function allows the user to add custom auto net assignment algorithms to the list in the Logic -> Auto Assign Net command's algorithms list in APD+. This list will always contain the Cadence standard algorithms (Router-Based, Nearest Match, and Constraint-Driven).

These names cannot be duplicate or overwritten by the customer.

Specifying a pair with either a duplicate algorithm or display name will cause the currently-existing (user) entry to be replaced.

### **Arguments**

<i>t_algorithm</i>	Text string (case sensitive) containing the name of the SKILL function to be called for this algorithm. The function must take two parameters:  First parameter: List of source pins.  Second parameter: List of destination pins.  The function should return "FAIL" if it was unable to complete due to some manner of code failure, or else should return a list of source pins that are still unassigned.
<i>t_displayName</i>	Text string (also case sensitive) that will be used as the name of this algorithm in the auto assign net command's pull-down menu for selecting the algorithm to use.

### **Value Returned**

*t* if algorithm successfully registered.

*nil* if registration failed (function not defined, name in use, etc).

## **axlGetWireProfileDirection**

```
axlGetWireProfileDirection(  
    profileName  
)  
==> "FORWARD"/"REVERSE"/nil
```

### **Description**

This function returns the direction of a wire profile definition defined in the database. Profiles may be either forward or reverse. If the profile definition name provided does not exist in the design, the return value will be `nil`.

### **Arguments**

*profileName*      The name of the wire profile to query the direction of.

### **Value Returned**

- "FORWARD" for forward-bond wire profile definitions.
- "REVERSE" for reverse-bond wire profile definitions.
- `nil` if the wire profile definition does not exist in the database.

## **axlGetAllVisibleProfiles**

```
axlGetAllVisibleProfiles()  
  ==> list of profiles / nil
```

### **Description**

Returns a list of all the bond wire profiles currently visible in the design.

### **Arguments**

Nothing.

### **Value Returned**

- list of profile names.
- nil if error or no profiles visible.

## **axlSetAllProfilesVisible**

```
axlSetAllProfilesVisible(visible)
  ==> t / nil
```

### **Description**

Turns all wire profiles in the design on or off.

### **Arguments**

*visible*                    *t* to turn all profiles on, *nil* to turn them off.

### **Value Returned**

*t/nil* to indicate success or failure.

## **axlGetBondWireLength**

```
axlSetBondWireLength (
    wire
    method
)
==> length/nil
```

### **Description**

Given a bonding wire object, this function returns the length of the wire, either as its 2D vector or its true 3D path.

### **Argument**

wire	dbid of bond wire
method	Method of length calculation. Values can be 3D or 2D.

### **Value Returned**

length	length in database unit
nil	if not a bond wire or method is not a valid value

## **axlLoadViaStructure**

```
axlLoadViaStructure(  
    t_vsName  
    t_filePath  
)  
==> o_vsDbid/nil
```

### **Description**

Searches for via structure XML/EXML files in `PADPATH` and load the symbol into the database.

**Note:** To delete unused symdefs use `axlDeleteObject`.

### **Argument**

<code>t_vsName</code>	Name of via structure symbol (in lower case).
<code>t_filePath</code>	Full path to the XML/EXML file.

### **Value Returned**

<code>dbid</code>	Of symbol definition
<code>nil</code>	Cannot find file, improper file syntax, the license level is not applicable, symbol with provided name already exists.

### **Example**

```
symdef = axlLoadViaStructure("VS_1" "./vs_1.exml")
```

### **See Also**

[axlDBCreateViaStructure](#)

## **axlSetBondWireProfile**

```
axlSetBondWireProfile(  
    bondWires  
    profileName  
)  
==> t/nil
```

### **Description**

This command allows you to change the bond wire profile on one or more bond wires in the design. The bond wire profile definition must already exist in the design.

**Note:** Changing a wire's profile changes not just the profile name, but also the wire's material and its diameter or width to match the new profile definition.

### **Argument**

bondWires	either a dbid or a list of dbids representing the bond wires to be modified.
profileName	the name of the new bond wire profile to assign to the bond wire(s).

### **Value Returned**

t	if profile is defined and one or more bond wires were modified.
nil	if an error occurred or no bond wires were modified.

**Note:** It is much more efficient to assign a wire profile to all bond wires, which should be set to that profile, in a single call to this function, than to call the function on each individual bond wire.

## **axlImportWireProfileDefinitions**

```
axlImportWireProfileDefinitions(  
    xmlFileName  
    setAsMaster  
)  
==> x/nil
```

### **Description**

This function will import the bond wire profiles defined in the XML file specified. If a profile is defined both in the XML file and in the current design, the design's definition will be updated to match the new definition being imported.

### **Argument**

*xmlFileName*

The name of the XML file on disk which includes the bond wire profile definitions to be read. If not in the current working directory, this should include the absolute or relative path to the XML file.

*setAsMaster*

If true, this XML file will be set as the master profile definitions file for this database. The master file is where the user can refresh profile definitions from if they are changed. Only one file is allowed to be the master.

### **Value Returned**

- *x*, the number of profile definitions successfully imported.
- *nil* if an error occurred (message printed to status window).

## **axlSetBondWireProfile**

```
axlSetBondWireProfile(  
    bondWires  
    profileName  
)  
==> t/nil
```

### **Description**

This command allows you to change the bond wire profile on one or more bond wires in the design. The bond wire profile definition must already exist in the design. Note that changing a wire's profile will change not just the profile name, but also the wire's material and its diameter or width to match the new profile definition.

**Note:** It is much more efficient to assign a wire profile to ALL bond wires which should be set to that profile in a single call to this function, than to call the function on each individual bond wire.

### **Argument**

<i>bondWires</i>	either a dbid or a list of dbids representing the bond wires to be modified.
<i>profileName</i>	the name of the new bond wire profile to assign to the bond wire(s).

### **Value Returned**

- *t*, if profile is defined and one or more bond wires was modified.

*nil* if an error occurred or no bond wires were modified.

## **axlSetDieStackData**

```
axlSetDieData (
    g_stackId
    s_dataType
    g_newValue
)
==> t/nil
```

### **Description**

This function sets the given data for the specified die.

**Note:** The command is available only in APD+.

### **Arguments**

<i>g_stackId</i>	Name or dbid of the die stack to get the data for.
<i>s_dataType</i>	Data type of the given value, one of the following: ' (DSA_CAVITY_DEPTH DSA_CAVITY_LAYER DSA_CAVITY_DIMENSION)
<i>g_newValue</i>	New value for the specified data type.

### **Value Returned**

*t* on success, otherwise *nil*

## **axlDBIsDieStackLayer**

```
axlDBIsDieStackLayer (
    t_layerName
)
==> t or nil
```

### **Description**

Verifies if layer is a die stack layer. This means that the attribute of the "paramLayer" parameter *dbid* called "type" has a value of "DIESTACK". This is normally used in the APD product.

### **Arguments**

t_layerName	Layer name "CONDUCTOR/<subclass>"
-------------	-----------------------------------

**Note:** CONDUCTOR is ETCH in Allegro PCB Editor.

### **Value Returned**

t	die stack layer
nil	If not.

### **See Also**

[axlDBIsDiePad](#)

### **Examples**

```
axlDBIsDieStackLayer("CONDUCTOR/TOP_COND") -> nil
```

## **axIGetDieData**

```
axlGetDieData (    g_dieId ) ==> die-data defstruct/nil
```

## Description

Gets the data for the given die and loads it into the a defstruct.

Only available in APD+.

## Arguments

`g_dieName` refdes or dbid of the given die.

## Value Returned

dieData	defstruct with data for the given die.
nil	Die does not exist or there was an error.

## ***defstruct fieldsExample***

dbid	dbid of die
refId	die refdes
memberType	DSA_DIE
stackName	parent die-stack name
stackPosition	integer position of member in stack
layerName	die pad layer
dieThickness	die thickness (flipchip bumps not included)
totalThickness	die thickness (flipchip bumps included)
stackHeightMin	starting height within stack
stackHeightMax	ending height within stack
origin	die symbol x/y location
rotation	rotation angle in degrees
extents	unioned extents of all members in stack
type	one of DSA_DIE_STANDARD or DSA_DIE_CODESIGN
attachType	one of DSA_DIE_FLIPCHIP or DSA_DIE_WIREBOND
orientation	one of DSA_DIE_CHIPUP or DSA_DIE_CHIPDOWN
bumpDiamAtPkg	flipchip bump diameter at package
bumpDiamAtDie	flipchip bump diameter at die
bumpDiamMax	flipchip bump diameter maximum
bumpHeight	flipchip bump height
bumpEcond	flipchip electrical conductivity w/units

```
data = axlGetDieData("FLIPCHIP_1")
printf("stack-pos = %L, layer-name = %L, attach-type = %L\n"
```

## **Allegro SKILL Reference**

### IC Packaging Commands

---

```
data->stackPosition  data->layerName data->attachType)  
  
==> stack-pos = 1, layer-name = "TOP_COND", attach-type = DSA_DIE_FLIPCHIP
```

## axlGetDieStackData

```
axlGetDieStackData (
    g_stackArg
)
==> stack-data defstruct/nil
```

### Description

Gets the data for the given die-stack and loads it into a defstruct.

Only available in APD+.

### Arguments

g_stackArg	name or <i>dbid</i> of the given die-stack
------------	--

### Value Returned

stackData	defstruct with data for the given die stack.
nil	Die stack does not exist or there was an error.

defstruct fields:

dbid	dbid of member
refId	member name
surface	one of DSA_SUBSTRATE_CAVITY_TOP, DSA_SUBSTRATE_CAVITY_BOTTOM
depth	DSA_SUBSTRATE_TOP, DSA_SUBSTRATE_BOTTOM, for cavity-placed dies, gives the number of layers below the surface the stack is placed
cavityClearance	For cavity-placed stacks, the clearance from the stack extents to the edge of the cavity on the lowest layer.
cavityExpansion	For cavity-placed stacks, the amount by which the cavity grows on each subsequent layer (the "width" of the stadium step).
stackHeightMin	starting height within stack
stackHeightMax	ending height within stack

rotation	rotation angle in degrees
extents	unioned extents of all members in stack

### **Example**

```
data = axlGetDieStackData("DIESTACK1")
printf("name = %L, minHeight = %L, maxHeight = %L\n"
data->name data->stackHeightMin data->stackHeightMax)

==> name = "DIESTACK1", minHeight = 0.0, maxHeight = 496.0
```

## **axlGetDieStackMemberSet**

```
axlGetDieStackMemberSet()  
  ==> list of die-stack member defstructs/nil
```

### **Description**

Returns a list of defstructs - one for each member of the given die stack.

Only available in APD+.

### **Arguments**

g_stackArg	name or <i>dbid</i> of the die stack
------------	--------------------------------------

### **Value Returned**

l_data	List of die-stack member data defstructs
nil	in case of an error

### **defstruct fields:**

dbid	<i>dbid</i> of member
refId	member name
memberType	one of DSA_DIE, DSA_SPACER, DSA_INTERPOSER
stackPosition	integer position of member in stack
layerName	etch object subclass of member
nextMemberOnSameLayer	flag indicating next member on same layer
prevMemberOnSameLayer	flag indicating previous member on same layer

### **Example**

```
data = axlGetDieStackMemberSet("DIESTACK1")  
foreach(member data
```

## **Allegro SKILL Reference**

### IC Packaging Commands

---

```
printf("refId = %L, memberType = %L\n" member->refId
      member->memberType)
)

==> refId = "FC1", memberType = DSA_DIE
    refId = "IPOSER_1", memberType = DSA_INTERPOSER
    refId = "WB1", memberType = DSA_DIE
    refId = "SPACER_1", memberType = DSA_SPACER
    refId = "WB2", memberType = DSA_DIE
```

## **axlGetDieStackNames**

`axlGetDieStackNames() ==> list of die-stack names/nil`

### **Description**

Returns a list of the names of all die stacks in the current design.

Only available in APD+.

### **Arguments**

None

### **Value Returned**

List of die-stack names or `nil` if none exist

## axlGetIposerData

```
axlGetIposerData(  
    g_iposerId  
)  
==> iposer-data defstruct/nil
```

### Description

This function fetches the data for the given iposer and loads it into a defstruct.



Only available in APD+.

### Arguments

g_iposerName	name or <i>dbid</i> of the given interposer
--------------	---

### Value Returned

iposerData	defstruct with data for the given iposer.
nil	iposer does not exist or there was an error.

defstruct fields:

dbid	<i>dbid</i> of interposer
refId	interposer ref-id
memberType	DSA_INTERPOSER
stackName	parent die-stack name
stackPosition	integer position of member in stack
layerName	etch-object layer (vias/clines/shapes)
totalThickness	dielectric + conductor thickness
stackHeightMin	starting height within stack
stackHeightMax	ending height within stack
origin	interposer symbol x/y location

rotation	rotation angle in degrees
extents	unioned extents of all members in stack
dielMat1	name of dielectric material used for substrate
dielThickness	thickness of dielectric material
condMat1	name of conductor material used for etch objects
condThickness	thickness of conductor material

### **Example**

```
data = axlGetDieData("IPOSER_1")
printf("stack-pos = %L, layer-name = %L, thickness = %L\n"
      data->stackPosition  data->layerName data->totalThickness)

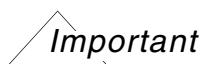
==> stack-pos = 4, layer-name = "IP1", thickness = 106.0
```

## axlGetSpacerData

```
axlGetSpacerData(  
    g_spacerId  
)  
==> spacer-data defstruct/nil
```

### Description

Gets the data for the given spacer and loads it into a defstruct.



Only available in APD+.

### Arguments

g_spacerName	name or <i>dbid</i> of the given spacer
--------------	---

### Value Returned

spacerData	defstruct with data for the given spacer.
nil	Spacer does not exist or there was an error.

defstruct fields:

dbid	<i>dbid</i> of spacer
refId	spacer ref-id
memberType	DSA_INTERPOSER
stackName	parent die-stack name
stackPosition	integer position of member in stack
layerName	etch-object layer (vias/clines/shapes)
totalThickness	dielectric + conductor thickness
stackHeightMin	starting height within stack
stackHeightMax	ending height within stack
origin	spacer symbol x/y location

rotation	rotation angle in degrees
extents	unioned extents of all members in stack
dielMatl	name of dielectric material used for substrate
dielThickness	thickness of dielectric material

## **Example**

```
data = axlGetDieData("SPACER_1")
printf("stack-pos = %L, layer-name = %L, diel-matl = %L\n"
      data->stackPosition data->layerName data->dielMatl)

==> stack-pos = 4, layer-name = "SP1", diel-matl = "PHENOLIC"
```

## **axlGetWireProfileColor**

```
axlGetWireProfileColor(t_profile)
==> color index / nil
```

### **Description**

This function will retrieve the color index associated with a bond wire profile. If no profile matching the name supplied is found in the database, nil is returned.

### **Arguments**

t\_profile                  Name of profile to retrieve color for.

### **Value Returned**

Color number assigned to profile if the profile is found.

nil if an error occurred or profile not found.

## **axlGetWireProfileVisible**

```
axlGetWireProfileVisible(t_profile)
==> t / nil
```

### **Description**

This function will retrieve the visibility status of a bond wire profile. If no profile matching the name supplied is found in the database, nil is returned.

### **Arguments**

t\_profile                  Name of profile to retrieve color for.

### **Value Returned**

t : if wire profile exists and is visible.

nil : if profile is invisible or does not exist.

## **axIPackageDesignCheckAddCategory**

```
axIPackageDesignCheckAddCategory (
    t_name
    t_bitmap
    t_description)
==> g_category / nil
```

### **Description**

This function will register a new category inside the *package integrity* command check tree of APD+.

You must define a category before adding checks to it. So, this function should always be called prior to [axIPackageDesignCheckAddCheck](#).

A newly added category will be inserted into the tree in alphabetically sorted order. Therefore, you do not need to manage the order categories are added by yourself.

**Note:** If the category name already exists, it will not be redefined.

### **Arguments**

<i>t_name</i>	Name of the category of checks as it should appear in the user interface. This name should be used when calling <a href="#">axIPackageDesignCheckAddCheck</a> to add specific checks.
<i>t_bitmap</i>	Name of the bitmap file which should be shown when this check category is active in the user interface. This should be a full path to the bitmap or else the bitmap must be resolvable through BMPPATH.
<i>t_description</i>	The description to be displayed in the GUI when this category is highlighted.

### **Value Returned**

<i>g_category</i>	SKILL defstruct defining the category.
-------------------	--

### **See Also**

[axIPackageDesignCheckAddCheck](#)

## Example

The following example add a new integrity check category for subsequent addition of user-defined rules:

```
axlPackageDesignCheckAddCategory(  
    "User-Defined Rules" "user_defined.bmp" "Descriptive Text"  
)  
==> g_category
```

The next time the Package Integrity Check tool is run, a new category will be shown, named "User-Defined Rules".

## **axlPackageDesignCheckAddCheck**

```
axlPackageDesignCheckAddCheck(  
    t_category t_name t_bitmap t_description  
    s_runCommand g_fixable  
) ==> defstruct defining check.
```

### **Description**

This function will register a new check in the specified category of the *package integrity* command check tree of APD+.

You must define a category before adding checks to it. So, this function should always be called after [axlPackageDesignCheckAddCategory](#).

A newly added check will be inserted into the tree in alphabetically sorted order. Therefore, you do not need to manage the order checks are added by yourself.

s\_runCommand is the SKILL function which should be called if this check is selected to run. This function MUST adhere to the following guidelines:

1. It must take exactly one argument, which is whether to fix errors it encounters or not.
2. It must return an integer value for how many errors were found in the database.
3. It must call the following functions:

```
axlPackageDesignCheck.LogError(<error string> <fixed>)
```

and

```
axlPackageDesignCheckDrcError(<error location> <dbids>)
```

to report any errors it finds.

These restrictions are imposed to ensure that output is consistent across all checks run by this command.

## Arguments

<i>t_category</i>	Name of the category this check should be placed under in the user interface tree. This should be the same name as sent to <a href="#">axlPackageDesignCheckAddCategory</a> .
<i>t_name</i>	Name of the check as it should appear in the user interface. This will be the name given to the check in the resulting log file, and will be the description for any external DRCs created.
<i>t_bitmap</i>	Name of the bitmap file that should be shown when this check is active in the user interface. This should be a full path to the bitmap or else the bitmap must be resolvable through <i>BMPPATH</i> .
<i>t_description</i>	The description to be displayed in the GUI when this check is highlighted. This description will also be printed to the log file ahead of any violations found for this check. As a result, the description should be as descriptive as possible in order to maximize its usefulness.
<i>s_runCommand</i>	A symbol representing the function to be called to check this rule. See <b>FUNCTION</b> description for details about the required format and return value of this function.
<i>g_fixable</i>	Boolean flag to tell the user on the interface whether problems found by this check can be automatically fixed or not.

## Value Returned

*g\_check* SKILL defstruct defining the check.

## Example

The following example adds a new integrity check to the "User-Defined Rules" category in the Package Integrity Check user interface.

```
axlPackageDesignCheckAddCheck(
    "User-Defined Rules" "My First Rule" "user_defined.bmp"
    "Descriptive Text" 'myCheckFunction t)
)
==> g_check
```

The next time the Package Integrity Check tool is run, "My First Rule" will be available under the "User-Defined Rules" category.

**See Also**

[axIPackageDesignCheckAddCategory](#), [axIPackageDesignCheckLogError](#),  
[axIPackageDesignCheckDrcError](#)

## **axlPackageDesignCheckDrcError**

```
axlPackageDesignCheckDrcError(l_location o_dbids)
==> nil
```

### **Description**

This function will create an external DRC marker for an error found by the currently running package integrity check. The tool itself will track the check being run so that it knows the name to use for the rule violation.

### **Arguments**

<i>l_location</i>	Location at which to place the DRC marker.
<i>o_dbids</i>	Optional list of database object ids that are associated with this error. Usually 0-2 objects are affected.

### **Value Returned**

nil

### **Example**

The following example creates a DRC marker about a missing via found with a fictional rule. In this case, since the via is missing, no dbids for objects are passed in:

```
axlPackageDesignCheckDrcError((0.0 0.0) nil)
==> nil
```

In the canvas, an external DRC marker is created at (0.0 0.0) with the name of the user-defined rule that generated the error.

### **See Also**

[axlPackageDesignCheckAddCheck](#), [axlPackageDesignCheck.LogError](#)

## **axlPackageDesignCheck.LogError**

```
axlPackageDesignCheck.LogError(  
    t_errorString  
    g_fixed  
    g_location)  
==> nil
```

### **Description**

This function will log an error found by this function to the log file if the log file is enabled. By using this interface, you are ensuring that the API will format your message consistently.

### **Arguments**

<i>t_errorString</i>	String to be printed to the log file. This should have all variable substitutions already done and be a simple string. This function will take care of any formatting necessary.
<i>g_fixed</i>	Boolean indicating whether the error was fixed by the tool.
<i>g_location</i>	Location where the error occurred, if applicable. This is appended to your log entry for you, and is used to let the user zoom to the error location in the design. If the location is unknown or not applicable, pass nil for the location.

### **Value Returned**

nil

### **Example**

The following example logs an error about a missing via found with a fictional rule. The error was not fixed by the caller:

```
axlPackageDesignCheck.LogError("Missing via" nil (0.0 0.0))  
==> nil
```

In the log file, the following is printed under the heading of the user-defined rule that found it:

```
"- Missing via at (0.0 0.0)"
```

**See Also**

[axIPackageDesignCheckAddCheck](#), [axIPackageDesignCheckDrcError](#)

## axlSetDieData

```
axlSetDieData(  
    g_dieId  
    s_dataType  
    g_newValue  
)  
==> t/nil
```

### Description

Sets the given data for the given die.



Only available in APD+.

### Arguments

g_dieName	name or <i>dbid</i> of the die to get the data for
s_dataType	data type of the given value, one of:  ' (DSA_BUMP_PACKAGE_DIAM_DBREP DSA_BUMP_PACKAGE_DIAM_STRING DSA_BUMP_DIE_DIAM_DBREP DSA_BUMP_DIE_DIAM_STRING DSA_BUMP_MAX_DIAM_DBREP DSA_BUMP_MAX_DIAM_STRING DSA_BUMP_HEIGHT_DBREP DSA_BUMP_HEIGHT_STRING DSA_BUMP_ECOND_STRING DSA_DIE_THICKNESS_DBREP DSA_DIE_THICKNESS_STRING)
g_newValue	new value for the specified data type.

### Value Returned

t on success, otherwise nil

## **axlSetDieType**

```
axlSetDieType(  
    o_componentDBID  
    t_dieType  
)  
==> t/nil
```

### **Description**

This function sets the attachment type for a die component to one of the available types. Currently, the supported attachment types are:

- WIREBOND
- FLIP CHIP

### **Arguments**

<i>o_componentDBID</i>	dbid handle of the die component to be configured.
<i>t_dieType</i>	Attachment type to configure this component as. Supported values are "WIREBOND" and "FLIP CHIP".

### **Value Returned**

<i>t</i>	if successful.
<i>nil</i>	if failed (non-die object passed or not packaging product).

### **Example**

```
axlSetDieType (myComp "WIREBOND")  
==> t
```

## **axlSetIposerData**

```
axlSetIposerData(  
    g_iposerId  
    s_dataType  
    g_newValue  
)  
==> t/nil
```

### **Description**

Sets the given data for the given iposer.

Only available in APD+.

### **Arguments**

g_iposerName	name or <i>dbid</i> of the given iposer
s_dataType	data type of the given value, one of:  ' (DSA_CONDUCTOR_MATERIAL DSA_CONDUCTOR_THICKNESS_DBREP DSA_CONDUCTOR_THICKNESS_STRING DSA_DIELECTRIC_MATERIAL DSA_DIELECTRIC_THICKNESS_DBREP DSA_DIELECTRIC_THICKNESS_STRING DSA_PART_NUMBER)
g_newValue	new value for the specified data type

### **Value Returned**

t on success, otherwise nil

## **axlSetSpacerData**

```
axlSetSpacerData(  
    g_spacerId  
    s_dataType  
    g_newValue  
)  
==> t/nil
```

### **Description**

This function sets the given data for the given spacer.



Only available in APD+.

### **Arguments**

g_spacerName	name or dbid of the spacer to get the data for
s_dataType	data type of the given value, one of:  ' (DSA_DIELECTRIC_MATERIAL DSA_DIELECTRIC_THICKNESS_DBREP DSA_DIELECTRIC_THICKNESS_STRING DSA_PART_NUMBER)
g_newValue	new value for the specified data type.

### **Value Returned**

t on success, otherwise nil

## **axlSetWireProfileColor**

```
axlSetWireProfileColor(t_profile n_color)
==> t / nil
```

### **Description**

This function will set the color of a wire profile to the given value.

### **Arguments**

t_profile	Name of profile to retrieve color for.
n_color	Color index to assign to the profile.

### **Value Returned**

t, if successful.

nil, if error (profile does not exist).

## **axlSetWireProfileVisible**

```
axlSetWireProfileVisible(t_profile g_visible)
==> t / nil
```

### **Description**

This function will make the identified wire profile visible or invisible.

### **Arguments**

t_profile	Name of profile to retrieve color for.
g_visible	t for visible, nil for invisible.

### **Value Returned**

- t, if successful.
- nil, if error (profile does not exist).

## **axlCDLNetlistSetValue**

```
axlCDLNetlistSetValue(  
    s_field  
    [g_value]  
    [b_locked]  
    [t_padStack]  
)  
⇒ nil
```

### **Description**

Configures a value in the CDL Netlist Out command of the `cdl out` command used to export CDL (Circuit Description Language). Use this function to automate the CDL Netlist Out form usually by also registering a trigger with the `axlTriggerSet()` command for the '`open`' event. Using this function along with the trigger, you can pre-configure the CDL Netlist Out form based on your requirements, the contents of the active design, and so on.

## Arguments

<i>s_field</i>	Name of the field to be configured. If <code>nil</code> , returns a list of all fields that are supported.
<i>g_value</i>	Value to set in the field. The value should match the type of value stored in the specified field. For example, for resistance, this is a numeric value.
	For the resistor checkbox column, this is a boolean.
<i>b_locked</i>	If passed, a boolean value indicates whether the field value can be modified or not.
	If true, the field gets locked from any interactive editing.
<i>t_padStack</i>	If configuring a row of the table in the form, for modifying padstack entry. <code>nil</code> sets the <i>All</i> rows.

## Value Returned

<code>t</code>	If successful, when configuring a value.
<code>nil</code>	Is passed for <i>s_field</i> , returns a list of fields, which may be set.

## Example

To set the name of the CDL netlist file to be written and prevent the user from modifying it:

```
; // File name should match current db name with extension of cdl.  
fileName = strcat(axlCurrentDesign() ".cdl")  
; // Prevent user from interactively editing value set by automation.  
axlCDLNetlist('outputFileName fileName t)
```

---

## User Interface Functions

---

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

Allegro 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. Allegro PCB Editor then calculates the placement location. An advantage of this method is that all windows automatically position themselves relative to the main Allegro 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)
```

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 Allegro PCB Editor. Allegro tools typically support three menus; drawing, symbol and shape. The Allegro command set is very different between these three design editors. Also menu sets exist for different tools such as APD+ and the SI (Signal Integrity) products.

All of the tiering (product levels) within a product are managed via the "#ifdef" statements within a single menu file. Typically, the settings of environment variables controlling the tiering are documented at the top of file.

**Note:** You cannot strip out the `#ifdef` statements to gain access to the missing commands.

Allegro finds the menus with its `MENUPATH` environment variable. You can find the default Allegro PCB Editor menu files in:

```
<cdsroot>/share/pcb/text/cuimenus
```



**As new products are added in a release, new menu files may be added.  
Cadence may change the name of any menu file in a release.**

## Allegro SKILL Reference

### User Interface Functions

---

The menus in this directory are as follows (due to the tools and software version you have loaded, some may not be present in your installation). You should not modify any other file type in this directory as only the menu files are supported for user modification.

**Table 10-1 Allegro PCB Editor Menu Files**

File Name	Description
allegro.men	Allegro PCB Editor menu for all Allegro PCB Editor .brd designs
pcb_symbol.men	Symbol menu for PCB products
partition.men	Partition menu for PCB products
specctragest.men	PCB SI menu
apd.men	APD+ menu
icp_symbol.men	APD+ symbol editor menu
apd_partition.men	APD+ Partition editor menu
apd_si.men	APD+ SI menu
padlayout.men	Pad Designer in the board graphics editor
padlaystn.men	Pad Designer (standalone)
allegro_free_viewer.men	Allegro Free Viewer
viewlayout.men	Allegro Viewer Plus

### Menu Terms

- Menu bar - the menu items seen at the top of a Window
- Menu item - a menu line; may either be a command, separator or a submenu.
- Separator - a horizontal line drawn to visually group menu items.
- Submenu - a pulldown (from the menu bar) or a pull-right (from another submenu). Submenus may only have a display and command association is not supported.

### Menu Design Considerations

- Certain dynamic items may exist. These are currently the MRU (most recently used files) and Quick reports.  
  
Do not attempt to modify these items.

- Do not use spaces in the display for the menu bar.
- All menu bar items should be submenus. Do not add a command menu item at this level.
- Do not add excessive items to the menu bar. If the menu bar displays on two lines for a typical window width you may have too many items.
- Keep the display text relatively short, especially on the menu bar.

#### MENU CUSTOMIZATION METHODS

- Provide your own customization menu via CDS\_SITE. Replace the Cadence provided menu (.men file) with your own.
  - Advantages: Relatively easy and no SKILL programming required.
  - Disadvantages: For new releases need to merge your menu changes with new Cadence menus. May need to modify multiple menus
- Overload your menu customizations on Cadence menus via SKILL [axlUIMenuRegister](#).
  - Advantages: Relatively easy with minimal SKILL programming. Depending on your site's additions may be immune to many Cadence menu changes.
  - Disadvantages: Cannot delete Cadence menu items or restrict your changes to a one Cadence menu.
- Register a axl menu Trigger notification via [axlTriggerSet](#).
  - Advantages: Almost as much flexibility as overriding the default menu file including targeting specific menus.
  - Disadvantages: Need to examine your SKILL code with new Cadence releases. Requires much more SKILL programming knowledge.

## 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` and `allegro_viewer`.

Tools support dynamically (via SKILL) modifying menus. For information, see [axlUIMenuFind](#).

## 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:

<b>Convention</b>	<b>Description</b>
[ ]	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 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. The command should be placed within a set of double quotes (""). Double quotes are not supported within this command string.

**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.

An #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 #ifndef is the negation of #ifdef.

## **Allegro SKILL Reference**

### User Interface Functions

---

```
>      environment variable is set. A #ifdef will cause 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, the #ifdef, #ifndef and #endif must
>      start at the first column of its line in the menufile.
<      The condition syntax supports multiple variables with OR '||' or
<      AND '&&' conditions. Also the negation character '!' is supported
<      for the variables:
```

## Allegro SKILL Reference

### User Interface Functions

---

These statements may be nested. The simple 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

<           # logically equivalent to above state using negation character
<           #ifdef !<env variable name>
<           [menu items which appear if the env variable is NOT set]
<           #endif
<
<           Also logical statements
<           1) if variable1 and variable2 are both set do the included statement
<               #ifdef <var1> && <var2>
<               [menu items which appear if both variables are set]
<               #endif
<
<           2) if either variable1 or variable2 is do the included statement
<               #ifdef <var1> || <var2>
<               [menu items which appear if either variable is 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 SKILL Reference

### User Interface Functions

---

#### **Example 3 - Simple Menu Example**

```
DISPLAY (indents reflect the various pulldown levels)
    FileHelp
        OpenContents
        ExportProduct Notes
            LogicKnown 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**

```
axlCancelOff(  
    )  
    ⇒ t
```

#### **Description**

Disables checking when a user clicks *Cancel*. See [axlCancelOn](#).

#### **Arguments**

None

#### **Value Returned**

Always returned t.

#### **Example**

See [axlCancelOn](#)

## **axlCancelOn**

```
axlCancelOn(  
    )  
    => t
```

### **Description**

Allows SKILL code to test for when a user clicks *Cancel*. When an operation is started that takes a long amount of time, enabling cancel allows a user to interrupt the operation by clicking on the Allegro traffic light or pressing the `Esc` key. The cancel checks are performed when the `axlCancelTest` function is called, allowing the SKILL code to determine the best time to abort an operation.

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 Allegro PCB Editor command line because Allegro 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**

Always returns `t`.

## See Also

[axlCancelSetFormClosable](#), [axlCancelClearFormClosable](#)

## Examples

```
count = 0
axlCancelOn()
while(count < 50000 && !axlCancelTest())
    ;; Do work here.  We will break out of the loop if
    ;; the user requests a cancel.
    count++
)
axlCancelOff()
```

## **axlCancelTest**

```
axlCancelTest(  
    => t/nil
```

### **Description**

See [axlCancelOn](#).

### **Arguments**

None

### **Value Returned**

Only `axlCancelTest` returns meaningful data.

t	User click cancel.
nil	User did not click cancel.

### **See Also**

[axlCancelOn](#), [axlCancelOff](#)

### **Example**

See [axlCancelOn](#)

## **axlCancelClearFormClosable**

```
axlCancelClearFormClosable(  
    o_form  
)  
⇒ t/nil
```

### **Description**

See [axlCancelSetFormClosable](#).

### **Arguments**

*o\_form*                  Form dbid

### **Value Returned**

t	If form is not closable during cancel action processing
nil	If not

### **See Also**

[axlCancelOn](#), [axlCancelOff](#), [axlCancelTest](#), [axlCancelSetFormClosable](#)

### **Examples**

See [axlCancelSetFormClosable](#)

## **axlCancelSetFormClosable**

```
axlCancelSetFormClosable(  
    o_form  
)  
⇒ t/nil
```

### **Description**

Allows a form to be closable during cancel checking.

When cancel processing is enabled by [axlCancelOn](#), most UI events are ignored. As a result any attempts to close a form during a cancel check are also ignored. These functions allow the caller to specify that the given form can be closed during cancel processing.

Closing the form when cancel is enabled will be treated as a user-initiated cancel request.

The [axlCancelClearFormClosable](#) call should be made before the form is closed.

### **Arguments**

*o\_form*                          Form dbid

### **Value Returned**

t	If form is closable during cancel action processing
nil	If not

### **See Also**

[axlCancelOn](#), [axlCancelOff](#), [axlCancelTest](#), [axlCancelClearFormClosable](#)

### **Examples**

```
; Create form, set form as active during cancel, then enable cancel.  
form = axlFormCreate((gensym) "my.form" '("C" "INNER") '_myFormCallback t)  
axlCancelSetFormClosable(form)  
axlCancelOn()  
  
;; Do work, check for cancel between iterations.
```

## **Allegro SKILL Reference**

### User Interface Functions

---

```
count = 0;
while(count < 50000 && !axlCancelTest()
    ;; Do something here
    count++
)

;; Unset form as active during cancel, disable cancel, close form.
axlCancelClearFormClosable(form)
axlCancelOff()
axlFormClose(form)
```

## **axlClipboardGetText**

```
axlClipboardGetText ()  
=> t_text/nil
```

### **Description**

This gets the current text in the system clipboard. Clipboard can contain data other than ASCII text in which case `nil` is returned.

### **Arguments**

No arguments

### **Value Returns**

<i>t_text</i>	Text in clipboard
<i>nil</i>	No text in clipboard

### **Examples**

```
axlClipboardSetText("hello world")  
text = axlClipboardGet()
```

### **See Also**

[axlClipboardSetText](#)

## **axlClipboardSetText**

```
axlClipboardSetText(  
    t_text/nil  
)=> t/nil
```

### **Description**

Sends indicated text to system clipboard. Only ASCII text is supported.

### **Arguments**

<i>t_text</i>	Text string. If <code>nil</code> or an empty string is specified, clipboard is emptied.
---------------	---

### **Value Returns**

<i>t</i>	Updated clipboard
<i>nil</i>	Failed

### **Examples**

See example section of [axlClipboardGetText](#)

### **See Also**

[axlClipboardGetText](#)

## **axICursorGet**

```
axICursorGet (   
    g_pixel  
) ==> l_xy
```

### **Description**

This command is used to obtain the current cursor position either in pixels (screen units) or converted into current design units. The mapping from pixels to design units takes into account the current window view and zoom factor of the design.

Accessing this in non-graphic mode is undefined.

### **Arguments**

<i>g_pixel</i>	If the value is set to <code>t</code> , the xy coordinates are specified in pixels. If the value is set to <code>nil</code> , current cursor position as it stands in current design is returned.
----------------	---

### **Value Returned**

The cursor position either in pixels (integer) or design units (floating point).

### **See Also**

[axICursorWarp](#), [axIUIControl](#)

## **axICursorWarp**

```
axICursorWarp (
    g_pixel
    l_xy
) ==> t/nil
```

### **Description**

Use this command to set the cursor position. May set the cursor either by pixel or design units. If setting by design units the new value must be within the current viewable window ([axIWindowBoxGet](#)).

**Note:** See [axICursorGet](#) for a discussion between pixel and design units.

### **Arguments**

<i>g_pixel</i>	If t return xy in pixels else return cursor position where it stands in current design.
<i>l_xy</i>	The xy values may be specified in pixel (g_pixel=t) or design units (g_pixel=nil)

### **Value Returned**

<i>t</i>	If moved cursor
<i>nil</i>	If bad arguments or moved cursor outside of main window.

### **See Also**

[axICursorGet](#), [axIWindowBoxGet](#)

## **axlMeterCreate**

```
axlMeterCreate(
    t_title
    t_infoString
    g_enableCancel
    [t_formname]
    [t_infoString2]
    [g_formCallback]
-> t/nil
```

### **Description**

Starts progress meter with optional cancel feature.

**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 <a href="#"><u>axlMeterIsCancelled()</u></a> .
<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> . <a href="#"><u>axlMeterIsCancelled</u></a> will also notice if a <i>Cancel</i> menu button has been pressed. If you do not give a form name <i>axlprogress.form</i> 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 fillings etc you may have on your form. This works the same as <i>g_formAction</i> in <a href="#"><u>axlFormCreate</u></a> .

## Value ReturnedSee Also

<i>t</i>	On success; otherwise nil.
----------	----------------------------

[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()
```

## **axIMeterDestroy**

`axIMeterDestroy() -> 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**

[axIMeterCreate](#)

## **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`.

[axlMeterCreate](#)

## **axIMeterUpdate**

```
axIMeterUpdate(  
    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: <ul style="list-style-type: none"><li>■ nil - leave info text as it is.</li><li>■ "" - clear info string field.</li></ul>
<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**

[axIMeterCreate](#)

## axlUIAppMode

```
axlUIAppMode (
    S_mode
    t_appCommand]
) -> g_return
```

### Description

Interacts with Allegro's application mode interface. Operation is driven by the first argument:

'inAppMode	Returns t if in application mode, nil in Idle mode
'mode	Returns application mode s a string, nil if in Idle mode
'commands	Returns the set of currently supported application modes. This depends upon the base product and sometimes the product options selected.
'none	Turns off application mode, Allegro is put in idle mode. Any dbids that are checked out remain active. Always returns t .
'set	Sets the application mode to string provided in argument 1. Any dbids are reclaimed (for example, dbid:removed). Returns t if app mode was set, nil if failure

### Arguments

<i>S_mode</i>	See above
<i>t_appCommand</i>	Only applicable with 'set

### Value Returned

<i>g_return</i>	Depends on <i>S_mode</i> see above 'inAppMode
-----------------	---

### Examples

- Get application mode commands
  - cmds = axlUIAppMode ('commands)
- To check if an application mode is active
  - amApp = axlUIAppMode ('inAppMode)

## **Allegro SKILL Reference**

### User Interface Functions

---

- Set application mode to none (for example, Idle)

```
axlUIAppMode ('none')
```

- Set application mode to general edit

```
axlUIAppMode ('set "generaledit")
```

## **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:** The intent of this procedure is to allow a custom menu to be loaded for debugging purposes.

### **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>e</code>	
<code>nil</code>	Menu not be located.

### **See Also**

[axlUIMenuFind](#)

## axlUIPopupMenu

```
axlUIPopupMenu (
  t_MenuFile
  [g_debug]
) ⇒ t_previousMenuName/nil
```

### Description

Dumps the current menu of the main window to the *t\_menuFile* file. 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.
[ <i>g_debug</i> ]	If this is 'print' then, only the commands contained in the menu file are dumped out. Allows for easier comparison between old and new menu files.

### Value Returned

<i>t_previousMenuName</i>	Full name of the file that is written.
<i>nil</i>	No file is written.

**Note:** The intent of this procedure is to provide a base menu file using which you can develop a customized menu.

## **axlUIColorDialog**

```
axlUIColorDialog(  
    r_window/nil  
    l_rgb  
) -> l_rgb/nil
```

### **Description**

Invokes standard color selection dialog box. You must provide a parent window, Allegro 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 nil, use main program window. Return handle of axlFormCreate 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.
nil	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  
    [s_level]  
)  
==> t
```

### Description

Displays the string *t\_message* in a confirm window.

The user must respond before any further interaction with Allegro 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.
<i>s_level</i>	Option level symbol; default is info level, other levels are warn and error.

### Value Returned

<i>t</i>	Always returns <i>t</i> .
----------	---------------------------

### Example

Inform user when a significant transition is being made:

```
axlUIConfirm( "Returning to Allegro. Please confirm." )
```

Alert user to an error:

```
axlUIConfirm( "Selected object has FIXED property." 'error' )
```

### See also

[axlUIPrompt](#), [axlUIYesNo](#), [axlUIYesNoCancel](#), [axlUIConfirmEx](#)

## **axlUIConfirmEx**

```
axlUIConfirmEx(  
    t_message  
    t_key/nil  
    [s_level]  
)  
==> t
```

### **Description**

Displays the string `t_message` in a confirmator window with an optional check box to never show the box again.

Functions same as `axlUIConfirm` except allows a check box to never show confirmator again. System remembers this selection so if user has indicated they do not want the box the call immediately returns.

Requires a unique `t_key` string which is used to remember the selection.

The optional `s_level` argument changes the info displayed to the user.

On program start/exit writes a file to `<HOME>/pcbenv/remember_<program>.txt`

### **Arguments**

`t_message`

Message string.

`t_key`

Unique key to remember user selection. If value of this parameter is nil, the command works like [axlUIConfirm](#).

`s_level`

Option level symbol; default is `info` level, other levels are '`warn`' and '`error`'.

### **Value Returned**

`t`: Always returns `t`

### **See Also**

[axlUIConfirm](#)

## Examples

Inform user when a significant transition is being made:

```
axlUIConfirmEx( "Use this command at your own risk." "mynewcommand")
```

## **axlUIControl**

```
axlUIControl(
    s_name
    [g_value]
)
==> g_currentValue/ls_names
```

### **Description**

Inquire about graphics canvas. Inquires and sets the value of graphics. If setting a value, the return is the old value of the control.

A side effect of most of these controls is if a form is active that is displaying the current setting it may not be updated. Additional side effects of individual controls are listed. Items will be added over time. Items currently supported:

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: vedge  
Value: (x\_x x\_y)  
Set?: No  
Description: Retrieves the virtual left top edge of the screen in pixels  
Equiv: none  
Side Effects: On UNIX always returns (0 0)

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.

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.

See above

## See Also

[axlOSControl](#)

## Examples:

Get screen size:

```
size = axlUIControl('screen)
-> (1280 1024)
```

Get pixel to user units:

```
axlUIControl('pixel2UserUnits)
-> 17.2
```

## **axlUIPopupMenuChange**

```
axlUIPopupMenuChange (
  x_menuId
  s_option
  g_mode
  ... <pairs of s_option/g_mode>
) -> t/nil
```

### **Description**

This changes one or more parameters of an existing menu item.

Unlike other menu commands this function can be safely done outside of the menu trigger callback if the menu command is associated with your SKILL code.

Changes allowed are a variable set of new value pairs:

**Table 10-2**

	s_option	g_mode
Enable/Disable menu	'enable	t/nil
Set/Unset Check mark	'check	t/nil
Change display text	'display	<new text display>
Change command text	'command	<new command string>

You should not attempt to change any separator menu items. Also do not attempt to assign command text to a submenu.

**Note:** See discussion in [axlUIPopupMenuFind](#) about menu changes.

## Arguments

x_menuId	The menuId from <a href="#">axlUIMenuFind</a>
s_option/g_mode pairs	See <a href="#">Table 10-2</a> on page 699

## Value Returned

t, if menu item is changed, and nil if the command failed to change the menu item.

## See Also

[axlUIMenuFind](#)

## Examples

- Set menu to be disabled

```
q = axlUIMenuFind( nil "add rect")
axlUIMenuChange(q 'enable nil)
```

- Enable and set check mark from previous example

```
axlUIMenuChange(q 'enable t 'check t)
```

## **axlUIMenuDebug**

```
axlUIMenuDebug(  
    [g_option]  
) => ll_menu/t/nil
```

### **Description**

A debug function for axl Menu Trigger. This helps debug issues with [axlUIMenuRegister](#).

### **Arguments**

g_option	data to query/clear
	'clear = clear the list of menus to load
	'list = return list of menus to be loaded (nil no menus)
	'trigger = clear the menu trigger callback and menus loaded

### **Value Returned**

t, call succeeded

nil, failed or if clear no menus present

ll\_menu, current list of menus queued

### **See Also**

[axlUIMenuRegister](#)

## **axIUIMenuDelete**

```
axlUIMenuDelete( x_menuId ) t/nil
```

## Description

This deletes a single menu item or submenu based upon what is the current find menu item.

**Note:** See discussion in [axlUIMenuFind](#) about menu changes.

## Arguments

x\_menuId the menuId from axlUIMenuFind

## Value Returned

`t`, if menu item is deleted else `nil` if failed to delete menu item

### **See Also**

### axIUIMenuFind

## Example

- Delete add rect command menu (add rect command is still available from command line)

```
q = axlUIMenuFind( nil "add rect")  
axlUIMenuDelete(q)
```

- Delete entire edit menu (assumes 2 menu item in menu bar)

```
q = axlUIMenuFind( nil 1)  
axlUIMenuDelete(q)
```

## axlUIMenuFind

```
axlUIMenuFind(  
    x_menuId/nil  
    t_cmdName/x_location  
    [g_menuOption]  
) ==> x_menuId/nil
```

### Description

Finds a menu item by location or a command. The location (`x_location`) is 0 based. The 0 location is the left or top most menu item. (Typically, this is the *File* menu item on the menu bar). A negative number may be used to specify a menu counting from the right side with a -1 indicating the menu furthest to the left or bottom.

Two modes are possible:

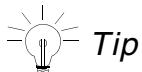
1. Find by name, finds menu item by command name.

This method cannot find menu bar items such as *File*. When finding by name you should pass `nil` as the first argument.

2. Find by `x_location`, identifies a menu item off the menu bar (`menuId = nil`) or submenu.

Menu searching is controlled via a menu stack. The first argument, `x_menuId`, controls the stack. For most operations, you should pass a `nil` to this argument. Typically, searching via the menu stack would use `x_location` as the second argument since the `t_cmdName` method is sufficient to find commands multi-levels deep in the menu hierarchy. If you have a nested search active then passing a `nil` will reset the stack. The stack is also popped if you provide a `menuId` older than the last id.

The `g_menuOption` when used in location mode returns the top or bottom of the indexed sub-menu (see below).



Examples shown below provide typical uses.



### ***CAUTIONS (release to release portability)***

- While not frequent, command names may change from release to release.
- Certain products or product tiers may not have a command.

## Allegro SKILL Reference

### User Interface Functions

---

- Menus may be reorganized so expecting to find a command on a particular submenu may not return the expected result in a new release.
- As always, adding Allegro commands or scripts to menus may require updates in a new release.
- See introduction of this section on menu recommendations.

## Arguments

x_menuId	menuId return of previous call or nil to search from menu root.
x_location	Find item by location. Location is 0 based. Therefore, the "File" menu is location 0. Negative numbers may be used where -1 is the right-most (or bottom-most) menu item.
t_cmdName	Find item by command name. This may not be just a command but is really a command line. For example, if the command is registered as "echo hello" then you must find by "echo hello" and not "echo".
g_menuOption	Permitted values are top or bottom.  If used with find by command returns the top or bottom of the menu where the command exists.  Bottom option also indicates to <a href="#">axlUIMenuInsert</a> to that a new menu item should be appended to end of the menu.  If used with find by location and the item is a submenu returns the top or bottom of that submenu.

## Value Returned

If successful returns a menu number else failure is indicated by a nil.

## See Also

[axlUIMenuInsert](#), [axlUIMenuChange](#), [axlUIMenuDelete](#), [axlUIMenuDump](#), [axlUIMenuLoad](#), [axlUIMenuRegister](#), [axlTriggerSet](#)

## Example

- To add to end of "Add" menu either of the following are equivalent (assumes add line exists on 3rd item of menu bar):

## Allegro SKILL Reference

### User Interface Functions

---

```
l = axlUIMenuFind(nil 3 'bottom)
l = axlUIMenuFind(nil "add line" 'bottom)
```

- Find Help menu, useful for adding a new sub-menu before the help menu

```
l = axlUIMenuFind(nil -1 nil)
```

- Find Top of Help menu, useful for adding new help menu items.

```
l = axlUIMenuFind(nil -1 'top)
```

- Find file menu

```
l = axlUIMenuFind(nil 0 nil)
```

- Find bottom of File – Import Menu

```
l = axlUIMenuFind(nil "load plot" 'bottom)
```

## **axlUIMenuInsert**

- Command to add menu item

```
axlUIMenuInsert(  
    x_menuId  
    t_display  
    t_command  
) -> t/nil
```

- Command to add Separator

```
axlUIMenuInsert(  
    x_menuId  
    'separator  
) -> t/nil
```

- Command to add Sub-menu

```
axlUIMenuInsert(  
    x_menuId  
    'popup  
    t_display  
) -> x_subMenuItemId/nil
```

- Command to add Sub-menu end (optional)

```
axlUIMenuInsert(  
    x_menuId  
    'end  
) -> t/nil
```

- Command to add multiple items

```
axlUIMenuInsert(  
    x_menuId  
    ll_items  
) -> t/nil
```

## **Description**

Inserts menu items to an existing menu. Several modes are supported:

1. Add a new menu item which dispatches a command when selected by user.
  - a. Add a new visual separator to menu.
2. Add a new sub-menu item. Assumption is that it will be populated by additional menu insert calls.

- a. End a sub-menu. This is optional, see menu stack discussion below.

**3. Add multiple menu items.**

This is implemented using a menu stack. [axlUIMenuFind](#) resets the stack and each submenu created increments the stack. The 'end' mode (submenu) decrements the stack. The menu stack allows the building of a menu tree with very little coding overhead. The stack depth is restricted to 8.



***Menu items should not be created outside a menu trigger. See discussion in [axlUIMenuFind](#). For development purposes you can create menu items outside of the menu trigger.***

## Arguments

x_menuId	menu id which can be obtained from <a href="#">axlUIMenuFind</a> or creating a submenu via this API. If nil uses the current menu on the menu stack
t_display	text that is shown in the menu. Possible values are: <code>separator</code> - add a separator (horizontal line) <code>popup</code> - create a new submenu
t_command	command to run <ul style="list-style-type: none"><li>■ this is ignored for a 'separator'</li><li>■ this is the display string for 'popup' option 'end'</li><li>■ pops the menu stack if creating a menu tree</li></ul>
ll_items	This is a list of t_display/t_command value pairs that instruct this interface to add multiple menu items and submenus in a single call. Both the 'separator' and 'end' options do not have to be a list.

## Value Returned

t - successful

nil - failed

x\_menuId - if creating a new submenu, the nesting id of new submenu

## See Also

[axlUIMenuFind](#)

## Example

- Add a separator before the add rect command

```
q = axlUIMenuFind( nil "add rect")
z = axlUIMenuInsert(q 'separator )
```

- Add a web link at the top of the help menu

```
q = axlUIMenuFind( nil -1 'top)
z = axlUIMenuInsert(q "Google" "http http://google.com" )
```

- Add a new submenu to the right of the help menu with two commands

```
q = axlUIMenuFind( nil -1)
; the nil is intention in here since it demonstrates
; the use of the current menu from find.
z = axlUIMenuInsert(q 'popup "MyMenu")
; the nil is required for the next 2 calls since we want to
; insert these items into MyMenu
z = axlUIMenuInsert(z "1" "echo hello 1" )
z = axlUIMenuInsert(z "2" "echo hello 2" )
```

- More nested menu

See <cdsroot>/share pcb/examples/skill/ui/menu.il

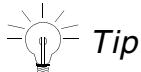
## **axlUIMenuRegister**

```
axlUIMenuRegister(  
    t_command/x_location  
    ll_menu  
    [g_menuOption]  
) => t/nil
```

### **Description**

This allows you to register menu items to be loaded when Allegro loads a new menu. It is a combination of [axlUIMenuFind](#) and [axlUIMenuInsert](#).

If more elaborate menu configuration is required consider calling [axlTriggerSet](#) directly.



*Tip*  
For registering menu items based upon product you need to use this API plus the [axlTriggerSet](#) method with one or more of the following APIs.

- To determine if the symbol editor is active use [axlIsSymbolEditor](#) function.
- [axlDesignType](#) may be used to differentiate between PCB or APD+ or SIP.
- Finally, [axlVersion](#) with an option:
  - 'programName' - another method to differentiate between PCB, APD+ or SIP
  - 'displayName' - differentiates between the products (PCB Designer vs Allegro Sigrity vs OrCAD PCB Designer Standard). This tends change considerably between releases and may change via ISR.
  - other options to this interface further classify products in a simple t/nil return classification.



- See [axlUIMenuFind](#) for cautions about portability across releases.
- When multiple menu registers are done, there may be dependencies. For example, if the first menu register adds a new submenu before the File menu the result will be not as expected if the second attempts to add a new item to the Edit menu via the location method.
- This API must never be called from within a [axlTriggerSet](#) callback function.

## Arguments

t_command	Command to insert menu before (see <a href="#">axlUIMenuFind</a> )
x_location	Location before to insert menu (see <a href="#">axlUIMenuFind</a> )
ll_menu	List of menu items to load (see format 3 option of <a href="#">axlUIMenuInsert</a> )
g_menuOption	Indication to add to top or bottom of menu (see <a href="#">axlUIMenuInsert</a> )

## Value Returned

t, if register function for indicated callback, nil, if the command failed to register trigger

## See Also

[axlUIMenuFind](#), [axlUIMenuInsert](#), [axlTriggerSet](#), [axlUIMenuDebug](#), [axlIsSymbolEditor](#)

## Example

See <cdsroot>/share pcb/examples/skill/ui/menu.il

## axlUIPopupMenuDump

```
axlUIPopupMenuDump (
  t_menuFile
  [g_debug]/t_menuState
  t_menuType
)
⇒ t_menuFile/nil
```

### Description

Dumps the current menus of the main window to the file *t\_menuFile*. The default menu filename extension will be added if *t\_menuFile* has none. There is no user interaction when an existing file is over-written.

### Arguments

<i>t_menuFile</i>	The name of the file, which the menu is to be dumped to. If <i>t_menuFile</i> is <i>nil</i> , the file name will be based on the program's default menu name (which may vary based on the current state of the program).
<i>g_debug</i>	If it is 'print then only the commands contained in the menu file are dumped. Allow for easier comparison between old and new menu files.
<i>t_menuState</i>	The state of the menu like enabled or disabled, checked or unchecked, which will be dumped into the file along with the name and command name of the menu
<i>t_menuType</i>	The type of the menu like parent menu contents or context menu contents, which will be dumped into the file.

### Value Returned

- The full name of the file that is written is returned.
- If no file is written, *nil* is returned.

**Note:** The intent of this procedure is to provide a base menu file from which a customized menu can be developed.

## axlUIPrompt

```
axlUIPrompt(  
    t_message  
    [t_default]/'password  
)  
==> 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 Allegro 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> .
<i>'password':</i>	Obscure and do not script user input.

### Value Returned

<i>t_response</i>	User response or default value.
<i>nil</i>	User selected <i>Cancel</i> .

## Example

```
axlUIPrompt( "Enter module name" "demo" )
⇒ "mymcm"
```

Prompts for a module name with a default `demo`. Typing `mymcm` overrides the default.

A text field displays, with the default value “`demo`”. To accept the default value, you may either press *Return* or select *OK*. Otherwise, type a new value in the text field and press *Return* or click *OK*. In this example, enter “`mymcm`” in the text field and click *Return*.

`axlprompt` returns the following:

```
==> "mymcm"
```

Password prompt:

```
ret = axlUIPrompt( "Enter password" 'password' )
```

## See also

[axlUIConfirm](#)

## **axlUIWCloseAll**

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

### **Description**

This closes all temporary windows (dialogs and text view windows). A temporary window is a dialog that closes if you open another design (for example, .brd). Via SKILL this window attribute is set by the axlUIWPerm API. The constraint manager is currently considered a permanent window but this may change in future releases. A blocking window (for example, File Browser dialogs) cannot be closed via this call.

### **Arguments**

None

### **Value Returned**

<i>t</i>	always
----------	--------

### **See Also**

[axlUIWPerm](#)

## **axlUIIconify**

```
axlUIIconify (
    r_window/t_window
    t/nil
) => t/nil
```

### **Description**

This command either creates an icon for a window or open a window from an icon. This is different from [axlUIWExpose](#), which also opens a window from an icon but exposes hidden windows and permits raising a window to the top of the stack. Note all sub-windows also open or closed to an icon. So, if you make the main Allegro PCB Editor window an icon, all of its child windows are also closed. *t\_window* name may change from release to release although this is not normal. *nil* may be used for the main window. Currently, Constraint Manager interface is not supported by this SKILL command.

### **Arguments**

<i>r_window</i>	Window ID.
<i>t_window</i>	Window name. This is the name that appears in PCB Editor scripting — invoked using the <code>setwindow</code> command.
<i>t</i>	To iconify window, <i>nil</i> open from an icon.

### **Value Returned**

<i>t</i>	For success
<i>nil</i>	For failure (The specified window could not be found) <sup>1</sup>

### **Examples**

Command to iconify Allegro PCB Editor

```
axlUIIconify("pcb")
```

### **See Also**

[axlUIWIsWindow](#), [axlUIWIsIconic](#), [axlUIWExpose](#)

## **axlUIWIsIconic**

```
axlUIWIsIconic(  
    r_window/t_window  
)=> t/nil
```

### **Description**

Is the window in an icon state. `nil` may be used for the main window. `t_window` name may change from release to release although this is not normal. Constraint Manager is not yet supported by this SKILL command.

### **Arguments**

<code>r_window</code>	Window ID.
<code>t_window</code>	Window name. This is the name that appears in Allegro scripting window — using the <code>setwindow</code> command.

### **Value Returned**

<code>t</code>	For success
<code>nil</code>	For failure (when the specified window cannot be found)

### **Examples**

Check if main window is in icon state

```
axlUIWIsIconic("pcb")
```

### **See Also**

[axlUIWIsWindow](#), [axlUIWIconify](#)

## **axlUIWIsWindow**

```
axlUIWIsWindow (
  t_window
)=> t/nil
```

### **Description**

Returns *t* if named window is open. *t\_window* name may change from release to release although this is not normal.

### **Arguments**

<i>t_window</i>	Window name. This is the name that appears in Allegro PCB Editor — using the <code>setwindow</code> command.
-----------------	--

### **Value Returns**

<i>t</i>	For success
<i>nil</i>	No window by that name is open

### **Examples**

Is Constraint Manager open

```
axlUIWIsWindow ("cmgr")
```

### **See Also**

[axlUIWClose](#)

## **axlUIWMove**

```
axlUIWMove (
    r_window/nil
    t_window
    l_xy
)
-> t/nil
```

### **Description**

Moves a window. New location (*l\_xy*) which is upper left corner, is specified in pixels.

*nil* may be used for the main window.

*t\_window* name may change from release to release although this is not normal.

**Note:** Constraint Manager is not yet supported.

### **Arguments**

<i>r_window</i>	Window ID or if <i>nil</i> the main window.
<i>t_window</i>	Window name. This is the name that appears in Allegro scripting from the setwindow command.
<i>l_xy</i>	( <i>x_X</i> <i>x_Y</i> )

### **Value Returned**

<i>t</i>	window moved
<i>nil</i>	Error, handle is not a window

### **See Also**

[axlUIWSize](#)

### **Example**

Move main window to upper left corner of the display.

```
axlUIWMove (nil 0:0)
```

## **axlUIWRedraw**

```
axlUIWRedraw(  
    r_window/nil  
)=> t/nil
```

### **Description**

Redraws indicated window. If window ID is `nil` redraws the main window.

### **Arguments**

*r\_window*                    Window ID or if `nil`, the main window.

### **Value Returns**

*t*                            For success

*nil*                        In case of failure (window already closed or not a window)

## **axlUIWSize**

```
axlUIWSize(  
    r_window/nil  
)  
-> ll_rect
```

### **Description**

Returns outer size of a window. Size is in pixels. x and y coordinates are upper left corner of window.

On UNIX/Linux, the y value will typically include an offset due to title bar height.

### **Arguments**

*r\_window*                    Window id or if *nil* the main window.

### **Value Returned**

*ll\_rect*                    ( (x\_X x\_Y) (x\_Width x\_Height) )

*nil*                        Error, handle is not a window

### **See Also**

[axlUIWMove](#)

## 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	<i>g_userType</i> is of type <i>r_windowMsg</i> .
nil	<i>g_userType</i> is not of type <i>r_windowMsg</i> .

### Example

```
logWindow =  
    axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)  
    axlIsViewFileType(logWindow)  
⇒ t
```

- Creates a window using `axlUIViewFileCreate` (See [axlUIViewFileCreate](#) on page 722.)
- Tests whether the window is a view file type.
- Returns t.

### See Also

[axlUIViewFileCreate](#)

## **axlUIViewFileCreate**

```
axlUIViewFileCreate(  
    t_file  
    t_title  
    g_deleteFile  
    [lx_size]  
    [lt_placement]  
    [g_formToExpose]  
)  
⇒ r_windowMsg/nil
```

### **Description**

Opens a file view window to display a file (*t\_file*), it is an error for file not to exist. Window should be given a title (*t\_title*).

If *g\_deleteFile* is set to *t*, the file is deleted when view window is quit or reused. It is suggested that applications not delete view files themselves as the Save and Print buttons will not work.

Size of viewable window is controlled by *lx\_size*. Default size is 24x80. Unpredictable results may occur for large row/column values.

Placement of window is handled by *lt\_placement* list. If this value is *nil*, the window is centered on editor.

Window may be deleted via program control via [axlUIWClose](#) function.

If a window of the same type with the same parent already exists and hasn't been pinned then the function try to reuse it automatically.

## Arguments

<i>t_file</i>	Name of the ASCII file to display. If the value is "" then last registered log file is displayed.
<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.
<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>	Window placement hints.  See the section on <a href="#">Window Placement</a> .
<i>g_formToExpose</i>	Optional handle of another window. If specified then this window is brought to the top of the desktop when the view file window is closed and also is the parent of this window. If not specified then the main program window is the parent.

## Value Returned

<i>r_windowMsg</i>	Window <i>r_windowMsg</i> .
<i>nil</i>	<i>r_windowMsg</i> not displayed.

## Example

- Displays the batch DRC log file, saving the window id.
- Deletes the file `drc.log` when the user exits the window.

```
logWindow = axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" nil)
```

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
  [g_formToExpose]
)
⇒ t/nil
```

### **Description**

Reuses the view window to display a file (*t\_file*). Error is thrown if the file does not exist. Window is given a title (*t\_title*).

Expects *r\_windowMsg* to be type of view window. If user quit the window it will re-open it at the old size/position.

File is deleted is *g\_deleteFile* is *t* when view window is quit or reused. It is suggested that applications not delete view files themselves as the Save and Print buttons will not work.

## Arguments

<i>r_windowMsg</i>	<i>dbid</i> of the existing view window created earlier with <code>axlUIViewFileCreate</code> .
<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.
<i>g_formToExpose</i>	Optional argument that defines the handle of a window to be exposed when the text file window is closed. Default is the parent set by <code>axlUIViewFileCreate</code> . Normally you should not use this argument.

## Value Returned

<i>t</i>	File displayed.
<i>nil</i>	File not displayed.

## Example

```
(axlUIViewFileReuse logWindow "ncdrill.log" "NC Drill Log" nil)
```

- Displays the file `ncdrill.log`, reusing the window `logWindow` created when displaying `batch_drc.log` in the `axlUIViewFileCreate` example.
- Exiting the window automatically deletes the file `ncdrill.log`.

## axlUIYesNo

```
axlUIYesNo (  
    t_message  
    [t_title]  
    [s_default]  
)  
==> 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. You must respond before any further interaction with Allegro PCB Editor.

### Note:

- If environment variable `noconfirm` is set, we immediately return *t* for yes and *nil* for no.

### Arguments

*t\_message*                  Message string to display.

### Value Returned

<i>t</i>	User responded <i>Yes</i> .
<i>nil</i>	User responded <i>No</i> .

### See Also

[axlUIConfirm](#)

### Examples

The following examples are a typical overwrite question.

## Allegro SKILL Reference

### User Interface Functions

---

```
axlUIYesNo( "Overwrite module?" )

axlUIYesNo( "Overwrite module?" nil 'no )

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) ? DfltrResponse(argv[2]) : MN_YES;

    return(MNYesNoWTTitle(str, title, dflt) ? ilcT : ilcNil);
}

/*
#endif DOC_C
```

## **axlUIWExpose**

```
axlUIWExpose(  
    r_window/nil  
)  
⇒ t/nil
```

### **Description**

Opens and redisplays a hidden or confined 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_window  
)⇒ t/nil
```

### **Description**

Closes a window, if it is open.

**Note:** Window may also be closed by user. See initial sections of the chapter for specific window types.

### **Arguments**

<i>r_window</i>	Window <i>dbid</i> .
<i>t_window</i>	Window name. This is the name that appears in Allegro scripting from the setwindow command.

*t\_window* name may change from release to release although this is not normal.

### **Value Returned**

<i>t</i>	Window closed.
<i>nil</i>	Window already closed, or <i>dbid</i> is not of a window.

### **Example**

- The following example, displays a window using axlUIViewFileCreate, and closes it using axlUIWClose.

```
logWindow = axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)  
;;; Other interactive code  
axlUIWClose(logWindow)
```

- Close Constraint Manager

```
axlUIWClose("cmgr")
```

## **Allegro SKILL Reference**

### User Interface Functions

---

#### **See Also**

[axlUIWIIsWindow](#)

## **axlUIWHelpRegister**

- Command to register new help file

```
axlUIWHelpRegister(  
    t_cmd  
    t_helpFile  
) -> t/nil
```

- Query if help file registered for command

```
axlUIWHelpRegister(  
    t_cmd  
) -> t_file
```

- Delete help file registered for command

```
axlUIWHelpRegister(  
    t_cmd  
    ""  
) -> t/nil
```

- Lists all cmds registered for help

```
axlUIWHelpRegister(  
    nil  
) -> lt_cmds
```

### **Description**

This registers a help document for a user written SKILL command or form (dialog). This is typically used in conjunction with axlCmdRegister. You should make this call at the time you do a axlCmdRegister instead of waiting until the SKILL code associated with the command executes.

You can also add the registrations via the `help_config.txt` file (see `<cdsroot>/share/pcb/help/help_config.txt`) placed at the site or pcbenv directory.

The document types (determined via file extension) supported on all platforms are:

- .txt - a plain text file displayed via Allegro's internal long message window
- .html - html browser displayed via a web browser
- .pdf - Acrobat file displayed by a Acrobat reader

On Windows other extensions are typically supported which are determined by what programs are installed on the computer (for example, doc for Word and ppt for PowerPoint).

## Arguments

<i>t_cmd</i>	Command name or form.<formname> for registering help for form buttons
<i>t_helpFile</i>	Document to display. Variable expansion is supported so you can embed Allegro env variables to make the installed location of the files relative to the variable setting.

## Value Returned

`t` for success, `nil` for failure (invalid arguments)

## See Also

[axlCmdRegister](#)

## Examples

Override add line help with contents of Allegro's env file

```
axlCmdRegister("add line" "$TELEENV")
```

## 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 `\e`), it is treated as a text (`\t`) message. (See [axIMsgPut](#) on page 946) 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**

<i>r_window</i>	Window <i>dbid</i> or, if <i>nil</i> , the main window.
-----------------	---

### **Value Returned**

<i>t</i>	Window is redrawn.
<i>nil</i>	<i>dbid</i> is not of a window.

## axlUIWBlock

```
axlUIWBlock(  
    r_window  
)  
⇒ t/nil
```

### Description



#### Caution

***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

<i>t_filename</i>	Name of file to edit.
<i>t_title</i>	Title bar name, or <code>nil</code> for default title bar.
<i>g_block</i>	Flag specifying blocking mode ( <code>t</code> ) or non-blocking mode ( <code>nil</code> ).

## Value Returned in Non-blocking Mode

<i>r_window</i>	Success
<code>nil</code>	Failure

## Value Returned in Blocking Mode

<code>t</code>	Success
<code>nil</code>	Failure

## **axlUIMultipleChoice**

```
axlUIMultipleChoice(  
    t_question  
    lt_answers  
    [t_title]  
)  
⇒ x_answer/nil
```

### **Description**

Displays a dialog box containing a question with a set of two or more answers in a list. You 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.
<i>t_title</i>	Optional title. If not present, a generic title is provided.

### **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.

### **Example**

```
ret = axlUIMultipleChoice("Pick a choice"  
    '("Pick me" "No Pick me" "I'm here!") "Cmd title")
```

## 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.



***With the html-based viewer the command is unable to return the number of items in scroll window. Return is only valid for the legacy text window.***

### 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 Allegro 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**

This has been mostly replaced by [axlUIWHelpRegister](#) that works for commands and forms.

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.

### **See Also**

[axlUIWHelpRegister](#)

## **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**

<i>o_childWindow</i>	Child window handle.
<i>o_parentWindow</i>	Parent window (if <i>nil</i> , 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**

<i>t</i>	Parent is successfully set.
<i>nil</i>	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**

<i>r_window</i>	The window id. If <i>nil</i> , signifies the main window.
<i>s_option</i>	One of the following: <ul style="list-style-type: none"><li>'<i>show</i>: Show and activate the window</li><li>'<i>showna</i>: Show but don't activate the window.</li><li>'<i>hide</i>: Hide the window.</li><li><i>nil</i>: Show available options.</li></ul>

### **Value Returned**

<i>t</i>	Window shown or hidden.
<i>nil</i>	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.

Points to be remembered while using the provided callback function.

- Processing in the callback should be relatively short in time
- Do not open or save the design
- Do not open or close forms or windows
- dbids may become stale
- `axlAddSimpleRbandDynamics` should not be used.
- Too many triggers active can impede performance.
- Allegro dbids are only valid within the callback. You cannot pass dbids in or out of this callback function. You always need to re-fetch them from the database.

## Arguments

<i>o_window</i>	The window the timer is associated with. If <i>o_window</i> is nil, the timer is associated with the main window.
<i>x_timeout</i>	Timeout in milliseconds before the timer is triggered and calls your callback procedure. Timeout is not precise because it depends on processing window messages.
<i>g_oneshot</i>	Controls how many times the timer triggers. Use one of these values: <i>t</i> - Timer goes off once and automatically removes itself. <i>nil</i> - Timer goes off at the set time interval continuously until it is removed by <code>axlUIWTimerRemove</code> .
<i>u_callback</i>	Procedure called when the timer goes off. Called with these arguments with its return value ignored: <i>u_callback</i> ( <i>o_window</i> <i>o_timerId</i> <i>n_elapsedTime</i> <i>o_window</i> : Window you provided to <code>axlUIWTimerAdd</code> <i>o_timerId</i> : Timer id which returned by <code>axlUIWTimerAdd</code> . <i>x_elapsedTime</i> : Approximate elapsed time in milliseconds since the timer was added.

## Value Returned

<i>o_timerId</i>	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. Do not 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.

## See Also

[axlUIWTimerRemove](#)

## Example

- **Basic:**

```
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.
)
```

- Other examples can be found at <*cdsroot*>/share pcb/examples/skill/ui/timer.il

## **axlUIWTimerRemove**

```
axlUIWTimerRemoveSet(  
    o_timerId  
)  
⇒ t/nil
```

### **Description**

Removes a timer added by `axlUIWTimerAdd`.

### **Arguments**

<code>o_timerId</code>	Id returned by <code>axlUIWTimerAdd</code> .
------------------------	--

### **Value Returned**

<code>t</code>	Timer removed.
<code>nil</code>	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 giving back the control to the UI message pump.

To use, make all your window changes and then make this call. If window ID is `nil`, exposes the main window.

### **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 <i>No</i> 1 for <i>Yes</i> 2 for <i>Cancel</i>
-----------------	--

### **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 '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 check box for the database LIBRARY_FIXED: Read-only check box 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.

## Allegro SKILL Reference

### User Interface Functions

---

*g\_arg*                    Generic argument passed through to *l\_callback* as the third argument.

#### Value Returned

*t\_name o\_dbid*        Selection was made and RETRIEVE\_OBJECT used.  
*t\_name nil*            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)
```

---

# Form Interface Functions

---

## Overview

This chapter describes the control types and functions you use to create Allegro PCB Editor forms (dialogs) and interact with users through them.

Allegro PCB Editor AXL forms support a variety of field types. See [Callback Procedure: formCallback](#) on page 858 and [Using Forms Specification Language](#) on page 767 for a complete description of field types.

The SKILL implementation of the forms package does not support the all functionality present in the core form package; short fields and variable tile forms.

### See Also

[axlFormCreate](#) - open a form

[axlFormCallback](#) - callback model for interaction with user

[axlFormBNFDoc](#) - Backus Naur Form, form file syntax, demos

[axlFormTest](#)

## Programming

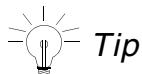
It is best to look at the two form demo.

- basic controls -- `axlform.il/axlform.form`
- grid control - `fgrid.il/fgrid.form`
- multi-select grid control - `fgrid-msel.il/fgrid.form`

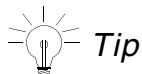
The first step is to create form file. Use `axlFormTest` to ensure fields are correctly positioned.

The following procedure is generally used.

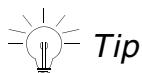
1. Open form (`axlFormCreate`)
2. Initialize fields (`axlFormSetField`)
3. Display Form (`axlFormDisplay`)
4. Interactive with user (`axlFormCallback`)
5. Close Form (`axlFormClose`)



*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.



*Tip*  
Use `axlFormTest("<form file>")` to interactively adjust of fields.



*Tip*  
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.

Multi-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 drop-down 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 `ax1FormSetDecimal`.

### ***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 check box

- a radio box

For both types the CHECKLIST control takes an argument for the text that should appear to the right of the checkbox.

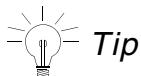
A radio box allows you to several checkboxes to be grouped together. The form package insures only one radio box be set. To enable a radio grouping provide a common text string as a third argument to the CHECKLIST keyword. An idiosyncrasy of a radio box is that you will be dispatched for both the field being unset and also for the field being set.

#### ***ENUM* (sometimes called combo box)**

Provides a drop-down to present the user a fixed set of choices. The drop-down 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 drop-down 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 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 hierarchical 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.

The length of step of the trackbar can be set in the form file where MIN is the tick mark interval and MAX is the length of the trackbar. The minimum tick mark is 1 and is usually indicated by setting MIN to 0 in the form file.

You can change the length and tick mark interval at run-time through [axlFormSetFieldLimits](#)

The trackbar indicator can be moved through [axlFormSetField](#).

### **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.

## Allegro SKILL Reference

### Form Interface Functions

---

You get information from the user using forms that support the following modes:

**Table 11-1 Form Modes**

<b>Form Mode</b>	<b>Description</b>
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><b>Note:</b> Use this programming model only with simple forms.</p>
Blocking with callback	<p>Prevents use of Allegro 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 Allegro PCB Editor forms. The user can work with both the form and other parts of Allegro PCB Editor.</p> <p>With Allegro 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 Allegro PCB Editor command starts by using <code>axlEvent</code>.</p>
Options form	<p>Allegro 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.</p>

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 Allegro PCB Editor core form package.

You can set background and foreground color on many form fields. For more information, see [axlFormColorize](#) on page 864. For information on color specific to grids, see [Using Grids](#) on page 781.

## 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 Allegro PCB Editor.
3. From the Allegro 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 Allegro 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 Allegro PCB Editor environment variable `TEL_SKILL` opens a SKILL interpreter window that is more flexible than the Allegro 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 Allegro 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 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 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 PCB Editor uses 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 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 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 |
INTSLIDE BAR 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, INTSLIDE BAR, MENUBUTTON, and
ENUMSET.

[MIN <value>]
[MAX <value>]
- Assigns a min and/or max value for the field.
- Both supported by field types: LONGFILLIN, INTSLIDE BAR,
REALFILLIN.
- Value either an integer or floating point number.
```

## Allegro 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 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 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.

## Allegro SKILL Reference

### Form Interface Functions

---

- 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 use the axlFormFlexDoc command to 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***

<b>Control</b>	<b>Resizing Options</b>
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.

FLEXMODE can have an optional pair of additional arguments that specify the minimum form width and height for flexing. The argument values are in character units. Flexing will stop in the given direction when the width/height goes below the specified value.

## 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 (`fx`, , `fw`, `fh`). 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 11 0 0 1 0.5  
FLEX g1 0 0.5 1 0.5  
FLEX 12 0 0 1 1
```

Form has a group (g1) containing a list (12). These are at the bottom of another list (11). Both lists share in any change of the form size. The second list (12) 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 11 0 0 1 1
```

Form has a group (g1) with a list member (11), 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 11 0 0 1 1  
FLEX 12 0 0 1 1
```

Form is a tabset (`t1`) with 2 tabs. Each tab controls a list (`11` and `12`) that accommodates the maximum change in the form size.

- Use `axlFormTest(<formname>)` to experiment with your form.

## Using Grids

You can use the axlFormGridDoc function to control 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.  
MULTISELROWAllows multi-row select (also set via Skill API,  
axlFormGridEvents)

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:

## Allegro SKILL Reference

### Form Interface Functions

---

3D - Display raised.  
NUMBER- For side header, display row number if application does not provide text.  
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

## Allegro SKILL Reference

### Form Interface Functions

---

In addition, the following standard form APIs may be used:

axlFormSetFieldVisible	Set grid visibility
axlFormIsFieldVisible	Is field visible
axlFormSetFieldEditable	Set grid editability
axlFormIsFieldEditable	Is field editable
axlFormBuildPopup	Change a popup
axlFormSetField	Set individual cell.
axlFormRestoreField	Restore last cell changed. Restore supports undoing last <i>change</i> event. Adding, deleting, or right mouse event reset restore.

#### ***Multi-row select support functions:***

axlFormGridSetSelectRow	control selection of rows
s	
axlFormGridSelectedCnt	number of rows selected
axlFormGridSelected	list of rows selected

## Data Structures

r_cell	User data type for cell update (see <a href="#">axlFormGridNewCell</a> on page 881)
r_formGridCol	Defstruct to describe column (see <a href="#">axlFormGridInsertCol</a> on page 876)

## 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.

TEXT	Column is composed of display only text.
STRING	Column supports editable text. See edit-combo.

## Allegro SKILL Reference

### Form Interface Functions

---

LONG	Column supports numeric data entry cells. See edit-combo.
REAL	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 871.)

By default, you create a grid with the 'rowselect' enabled which is typically appropriate for a multi-column table.

## **Multi-row Selection**

A super-set of row selection is the multi-row selection option. With this option the user can select multiple rows. Grids running in this mode do now support cell select or change options.

This is set in SKILL via:

```
axlFormGridEvents (<form> <grid> '(mrowselect))
```

or from the formfile by adding the `MULTISELROW` option to the grid's `OPTION` line.

Standard selection model is supported (not extended). This means:

- left click selects a row
- shift-left click selects all rows between the initial and current row
- ctrl-left click on to selection of row that is currently selected, it de-selects
- control-a selects all rows

APIs are provided (see above) to get current selected rows and set or clear row selections.

Finally, since multiple rows may be selected the standard form callback mechanism only informs you of a selection event. You need to utilize `axlFormGridSetSelectRows` to determine the current selection.

## **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>
```

## Allegro SKILL Reference

### Form Interface Functions

---

- 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.

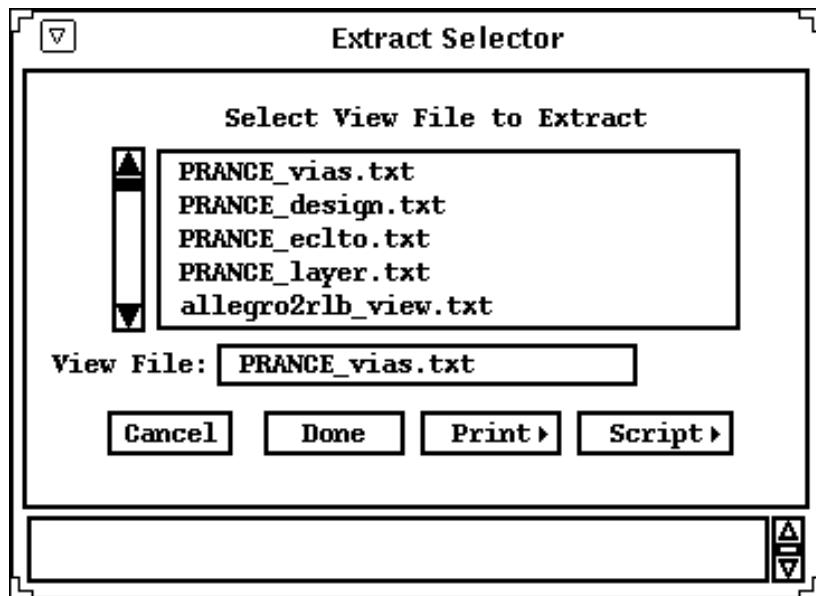
See [Using Grids](#) on page 781 for a grid overview.

## 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 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 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
    cdr( 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
```

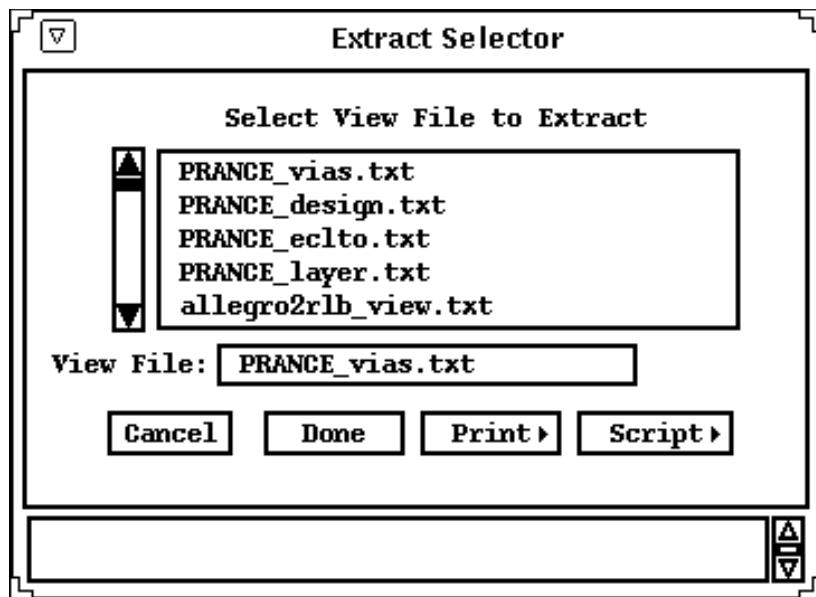
- Creates a form named `form` with the callback function `_formAction` that analyzes user action stored in `form->curField` and responds appropriately.

## Allegro SKILL Reference

### Form Interface Functions

- 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 Allegro 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 *Done* button. Closes the form, clears input using `axlCancelEnterFun`, and calls the `_extract` function to execute the data extract.

"cancel"                        The user selected the *Cancel* button. Closes the form, clears input using `axlCancelEnterFun`, and calls the `_extract` function to execute the data extract.

## Allegro SKILL Reference

### Form Interface Functions

---

"view_file"	The user selected the <i>View File</i> field, possibly typed an entry, and pressed <i>Return</i> . Sets the <i>view file</i> name to the current value of the <i>View File</i> 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 <i>form-&gt;curValue</i> , and the program sets <i>selectedFile</i> (the name of the currently selected extract file) to the new value, and displays it in the <i>View File</i> field.

The *Print* and *Script* buttons have pop-ups that call predefined Allegro 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".
TILE
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"
```

## Allegro SKILL Reference

### Form Interface Functions

---

```
ENDFIELD  
ENDTILE  
ENDFORM
```

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"
```

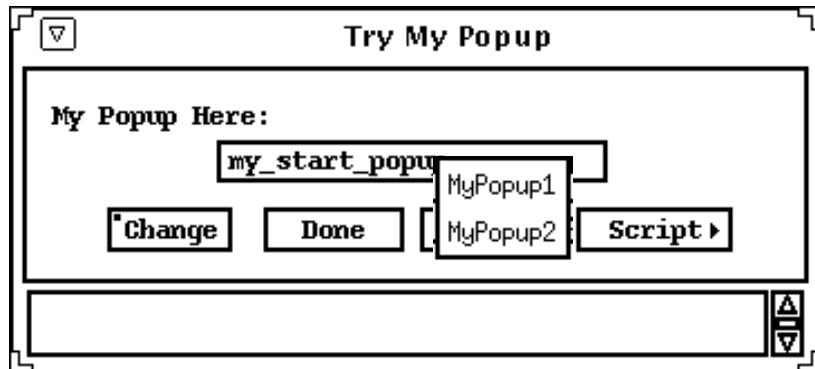
## Allegro SKILL Reference

### Form Interface Functions

```
        "MyPopNValue2"))
axlFormSetField(form "my_popup"
               "My Third Popups")
)
)
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`.

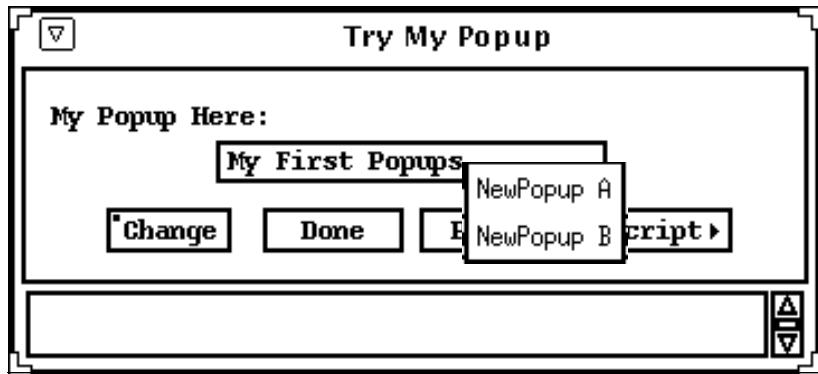


3. Click *Change*.

## Allegro 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.

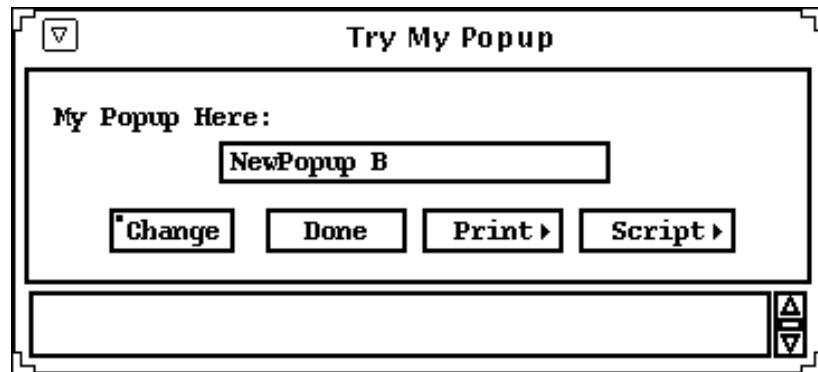


4. Make a selection.

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.

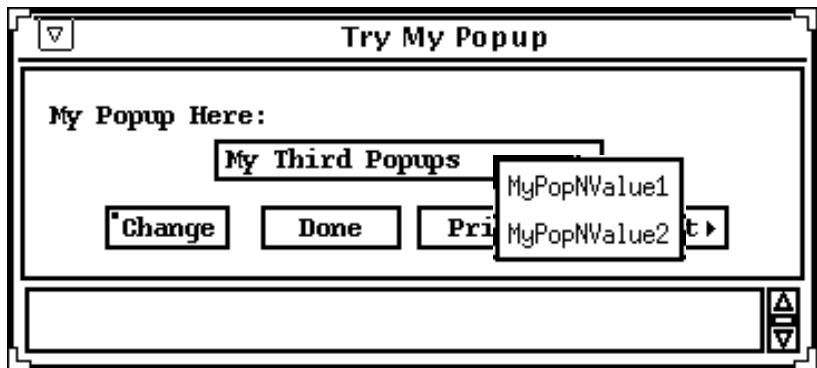


5. Click *Change*.

## Allegro 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 grammar 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.

- popups work slightly differently when applied to fillin versus other supporting fields, such as, ENUMs and BUTTONNs. With Fillin fields, such as, Strings and long, the display portion is always sent back to the application while other supporting field types, such as, ENUMs, send the dispatch portion.

The same applies with setting the field. For ENUMs, you must call the form API with the dispatch value while fillins expect the display string.

### ***message\_def***

MESSAGE messageLabel messagePriority "text".

### ***form\_options***

#### [TOOLWINDOW]

- this makes a form to be a tool window 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 PCB Editor/APDuses mostly variable width while SpectraQuest (Allegro PCB SI)and  
SigXplorer use fixed width fonts.

Note: This option is obsolete in the 17.4 release.

#### [AUTOGREYTEXT]

- when a fillin or enum control is greyed, grey static text to the left of it.

#### [NOFOCUS]

- when a form opens focus is set to form window which allows you to immediately enter data. If you create a form with no editable fields or don't wish to have the form grab focus set this option.

#### [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.

## Allegro SKILL Reference

### Form Interface Functions

---

#### ***tile\_def***

```
TILE [<titleLabel>]      (4)
[TPANEL tileType]
{{text_def}}
{{group_def}}
{{list_def}}
{{field_def}}
{{button_def}}
{{spacer_def}}
{{grid_def}}
{{flex_def}}
ENDTILE
```

#### ***tabset\_def***

```
TABSET [label]
[OPTIONS tabsetOptions]
FLOC x y
FSIZE w h
{tab_def}
ENDTABSET
```

#### ***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
```

***field\_def***

```
FIELD label  
FLOC x y  
[FSIZE w h]      (8)  
[HELPTIP "tip"]    (12)  
field_type  
field_options  
ENDFIELD
```

***button\_def***

```
FIELD label  
FLOC x y  
[FSIZE w h]      (8)  
MENUBUTTON "display" w h  
[OPTION button_options  
[button_bitmap]
```

## Allegro SKILL Reference

### Form Interface Functions

---

#### ***spacer\_def***

```
SPACER  
FLOC x y  
[FSIZE w h]      (14)  
ENDSPACER  
ENDFIELD
```

#### ***grid\_def***

```
GRID fieldName  
FLOC x y  
FSIZE w h      (8)  
[OPTIONS INFO | HLINES | VLINES | USERSIZE | MULTISELROW ]  
[POP "<popupName>"]  
  
[GHEAD TOP|SIDE]  
[HEADSIZE hlw]  
[OPTION 3D|NUMBER|MULTI]  
[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***

The OPTIONS line permits multiple options

## Allegro SKILL Reference

### Form Interface Functions

---

[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]

- 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.

## Allegro SKILL Reference

### Form Interface Functions

---

[OPTIONS password]

- string type only
- Field contains a password. Mask text.

[OPTIONS space]

- string type only
- preserves leading and trailing white space. By default this is stripped.

[OPTIONS dropfile]

- string and multiline types only
- Allows a file to be dropped into the field (Windows drag and drop)  
Shortcuts are not resolved.

#### *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.

***button\_bitmap***

[BITMAP [<bitmapFile>|#<resource>] ]

- display a bitmap for this button.

***dispatch***

- string that is dispatched to the code.

***display***

- string that is shown to user

***bitmapFile***

- name of a bmp file. Can be found via BMPPATH. Bitmap should be static.  
Animation should not be used.

- #<resource> only available for Cadence applications. It is obsolete.

***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.

[INVISIBLE]

field by default is not displayed

[STRETCH]

- THUMBNAIL field type.
- stretch bitmap to fit thumbnail rectangle, default is center bitmap.

[MAP3DCOLORS]

THUMBNAIL field type.

- search the color table of the .bmp and replace the following shades of gray with corresponding 3D color:

dk gray RGB(128,128,128) - COLOR\_3DSHADOW

gray RGB(192,192,192) - COLOR\_3DFACE

lt gray RGB(223,223,223) - COLOR\_3DLIGHT

This option is typically used to blend the .bmp's background color with the user's dialog background. If using this option, don't reserve these 3 gray colors for background only.

### ***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.

### ***gridOptions:***

Several of these options can also be controlled at run-time via UIFGridVarEvents or UIFGridVarOptions:

- INFO - grid is for info only; can be scrolled but items cannot be selected
- HLINES - display horizontal separator lines
- VLINES - display vertical separator lines
- USERSIZE - user can resize columns
- MULTISELROW - grid is opened in multi-select row mode

### ***gHeadOptions:***

- 3D - display header in 3d mode
- NUMBER - auto-number side header (default is blank)
- MULTI - display top header as multiple lines (default single line with clipping)

### ***tileLabel***

- name used to allow code to refer to this tile.
- should use lower case.

## Allegro SKILL Reference

### Form Interface Functions

---

- 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.

***flex\_def*** - rule based control sizing upon form resize (see [axlFormFlex](#))

[FLEXMODE <autorule> [minWidth minHeight]]  
[FLEX <label> fx fy fw fh]

FLEXMODE <autoRule> [minWidth minHeight]

FLEX fx fy fw fz

- see [axlFormFlexDoc](#)

***autorule*** - generic sizing placement rule

fx  
fy  
fw  
fh

- floating value between 0 and 1.0

#ifdef:

#ifndef:

#else:

#endif:

- Conditionally read portions of the form file based upon the settings of Allegro environment variables
- These statements may be nested.

## Allegro SKILL Reference

### Form Interface Functions

---

- Note the negation character '!' was added in 15.7. Forms using this capability will not function correctly in earlier releases.

Use #ifdef/#endif and #ifndef/#endif to make items conditionally appear in the menu depending on whether a specified environment variable is set.

An #ifdef causes the form item(s) to be ignored unless the environment variable is set. You must have one #endif for each #ifdef or #ifndef to end the block of conditional menu items. Also, the #ifdef, #ifndef and #endif must start at the first column of its line in the formfile. The #ifndef is the negation of #ifdef.

The #else statement may be inserted between the #if/#endif statements.

The condition syntax supports multiple variables with OR '||' or AND '&&' conditions. Also the negation character '!' is supported for the variables:

The simple syntax is:

```
#ifdef <env variable name>  
[form items which appear if the env variable is set]  
#endif  
  
#ifndef <env variable name>  
[form items which appear if the env variable is NOT set]  
#endif  
  
# logically equivalent to above state using negation character  
#ifdef !<env variable name>  
[form items which appear if the env variable is not set]  
#endif  
  
#ifdef <env variable name>  
[form items which appear if the env variable is set]  
#else  
[form items which appear if the env variable is set]
```

```
#endif
```

Also logical statements:

1) if variable1 and variable2 are both set do the included statement

```
#ifdef <var1> && <var2>  
[form items which appear if both variables are set]  
#endif
```

2) if either variable1 or variable2 is do the included statement

```
#ifdef <var1> || <var2>  
[form items which appear if either variable is set]  
#endif
```

**Notes:**

- 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 definition 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.
- FSIZE - 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.

- 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 option. When not set (the default) the drop-down is only presented under user control. When height greater than 1 then the drop-down 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.
  - HELPTIP supports two modes for dynamic help association with fields.
    - Default mode requires that you define an INFO field with the label helptip somewhere on your form. For tabbed forms, an INFO helptip field may be added to each tab for better localization of help information. When field is hovered over, the assigned helptip will be displayed in the INFO field.
    - Add the TOOLTIPHELP option somewhere in the form. If set, help tip information will be displayed as tooltips directly with the field when mouse is hovered over. Tooltip based help supports full HTML content, allowing the embedding of graphics and other data to provide additional context information for your user. For HTML tags, multiple HELPTIP lines may be defined for the same form field for easier HTML formatting. Do not enclose each line HTML help tip information in "s.
  - Options should appear after the fieldtype and MULTILINE in button type.
- 

The forming syntaxes are NOT supported by the form editor.

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
{} }
```

## **Allegro SKILL Reference**

### Form Interface Functions

---

- FSIZE is optional for SPACER fields, and will default to 0,0 (both vertical and horizontal spacing). If set, a non-zero value indicates a fixed amount of space to be reserved. A zero value indicates the spacer should expand in this direction to keep fields in place on the sides of it.

## **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. 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.
------	---

## Allegro SKILL Reference

### Form Interface Functions

---

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>

#### Notes:

- What distinguishes between a radio button and check box is that radio buttons are a group of check boxes where only one can be set. To relate several check boxes as a set of 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 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 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 three methods:
  - Static bitmap declared via form file.
  - Bitmaps that can by changed by the application at run-time.
  - Basic drawing canvas (see `ax1GRPDoc`).
- Buttons are stateless. The application cannot set the button to the depressed state. You can only use `ax1FormSetField` 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.

## Allegro SKILL Reference

### Form Interface Functions

---

**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, otherwise see bullets 4 and 5 in the following <b>Notes</b> section.
row	no	integer	Grid control only.
col	no	integer	Grid control only.
treeViewSelState	no	integer	Tree control only.

**Notes:**

- 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 that can be changed by the program at the run-time. All other labels appear in the fields list and can be changed by the user (even buttons, tabs, grayed 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/(t_formName t_contents)/(t_formName (t_contents))  
    [1t_placement]  
    g_formAction  
    g_nonBlock  
    [g_stringOption]  
)  
⇒ r_form/nil
```

### Description

Creates a dialog based on the form descriptive file *t\_formfile*. This call only supports forms of type "fixed" and fails if *t\_formfile* contains any variable tiles. This function does not display the form. Use `axlFormDisplay` to display a form.

An alternative interface is supported that allows embedding the contents of the form file in the SKILL code. Instead of passing the external form file name provide the name (*t\_formName*) for scripting purposes and form file contents (*t\_contents*) as string. The packaged SKILL code has a example of this method at the end of the `<cdsroot>/share/pcb/examples/form/finline.il` file. This method has the advantage of only distributing one file.

Rules to remember when creating this form content string:

- Every non-blank line must have a tab character

Example:

```
FILE_TYPE=FORM_DEFN VERSION=2
```

- Any embedded quotes must be escaped (use backslash '\')

Example:

```
MENUBUTTON \"Ok\" 10 3\n
```

- Any parenthesis '(' must be escaped '\'

**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.

## Allegro SKILL Reference

### Form Interface Functions

---

#### Arguments

<i>s_formHandle</i>	Global SKILL symbol used to reference form. <b>Note:</b> 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. <i>axlFormCreate</i> uses the Allegro PCB Editor environment variable, FORMPATH, to find the file, if <i>t_formfile</i> is not a full path. The filename, by convention, should use the .form extension. Alternative interface to embed form file into SKILL code: <ul style="list-style-type: none"><li>■ <i>t_formName</i>: Name of form (used for scripting)</li><li>■ <i>(t_contents)</i>: Contents of form file. This may be a string or a list containing or a string. The string format is obsolete and you should use <i>t_contents</i></li><li>■ <i>t_contents</i>: the list with string format</li></ul>
<i>lt_placement</i>	Form placement. Allegro PCB Editor uses its default placement if this argument is <i>nil</i> . See <a href="#">Window Placement</a> on page 663
<i>g_formAction</i>	Specifies the SKILL commands (callbacks) to be executed after every field change (Note that this is very different from Cadence IC forms). You can set this to one of the formats shown.
<i>g_form</i>	Action 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 <i>axlFormCreate</i> as its only parameter.)
<i>nil</i>	<i>axlFormDisplay</i> blocks until the user closes the form. You must place a <i>Done</i> button (field name <i>done</i> ) and optionally a <i>Cancel</i> button (field name <i>cancel</i> ) 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.

## Allegro SKILL Reference

### Form Interface Functions

---

<i>g_nonBlock</i>	If <i>g_nonBlock</i> 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 ( <i>g_formAction</i> ) is nil, then axlFormDisplay ignores <i>g_nonBlock</i> , and the form runs in blocking mode.
	Use of blocking mode blocks the progress of the SKILL code, but does not prevent other Allegro PCB Editor events from occurring. For example, if blocked, users can start the <i>Add Line</i> command from Allegro PCB Editor menus.
<i>g_stringOption</i>	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

<i>r_form</i>	<i>dbid</i> of form created.
nil	No form created.

### Example

See *<cdsroot>/share pcb/examples/form*

- basic: demonstrates basic form capabilities
  - finline.il shows correct inline method
  - grid: demonstrates grid control capabilities
  - wizard: form when used in a Wizard mode
  - finline: demonstrates inline option to avoid having a .form file

See [AXL Forms: Example 1](#) on page 789.

### See Also

- [axlFormIntroDoc](#): Introduction to the Allegro Form Package.
- [axlFormBNFDoc](#): Form file language description
- [axlFormCallback](#): Methods and structures for interacting with user.

## **axlFormClearMouseActive**

```
axlFormClearMouseActive(  
    r_form  
)  
==> t/nil
```

### **Description**

Clears the option to dispatch the MouseActive event on a form.

### **Arguments**

*r\_form* Handle for the form

### **Value Returned**

t	Option was cleared
nil	<i>r_form</i> does not reference a valid form

## **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, Allegro 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 789:

```
(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 *dbid*.

### **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 789.

```
axlFormDisplay( form)
```

## axlFormBuildPopup

```
axlFormBuildPopup(  
    r_form  
    t_field  
    l_pairs  
)  
⇒ t/nil
```

### Description

This provides the ability to dynamically change popups of fields that have them. These fields are enum (or pop-up) and other fields that have a popup icon. Buttons, optionally, may also have a popup if they have a right arrow. Attempting this call on a field without a popup is an error.

### Arguments

<i>r_form</i>	a form handle
<i>t_field</i>	Name of form field.
<i>l_pairs</i>	May be one of four formats where each element is a single popup entry. A maximum of 256 popup entries are allowed. <ul style="list-style-type: none"><li>■ normal<ul style="list-style-type: none"><li>( (t_display t_dispatch) ... )</li></ul></li><li>■ alternative normal<ul style="list-style-type: none"><li>( t_displayNdispatch ... )</li></ul></li><li>■ for enum field types<ul style="list-style-type: none"><li>( (t_display x_dispatch) ... )</li></ul></li><li>■<ul style="list-style-type: none"><li>( (t_display t_dispatch/x_dispatch options) ... )</li></ul></li></ul> Options can be 1 or 2 additional list options that are S_color/x_color for enum field types with color; bold or underline for bold or underlined items.

**Note:** All entries in an *l\_pairs* argument must be the same type of format. That is, you cannot have a list containing, for example, both display/dispatch strings and display/enum types, or display/dispatch and single-string entries. Must be one of the

## Allegro SKILL Reference

### Form Interface Functions

---

formats described. Each list object defines a single popup entry.

**Table 11-3** `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-&gt;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 SKILL Reference

### Form Interface Functions

---

**Table 11-3** l\_pairs Format Options

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))</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 1000 pop-up entries in one pop-up.
- If creating a dynamic popup (entries created under program control) a dummy entry must exist in the form file or build popup will fail. Example:  
  
`<popupname> """".`
- The field name is actually a search mechanism. We first search the fields for the field name with a popup and then search the popup names. Since the only way to change grid

## **Allegro SKILL Reference**

### Form Interface Functions

---

column or cell based popups is by popup name you may run into failures if that popup name has the same name as another field in the form.

#### **Value Returned**

t	Field set.
nil	Field not set.

#### **Example**

See [AXL Forms: Example 2](#) on page 793.

## **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**

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of field.

### **Value Returned**

<i>g_value</i>	Current value of the field.
<i>nil</i>	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 789.
2. Enter the command `myExtract()` on the SKILL command line.

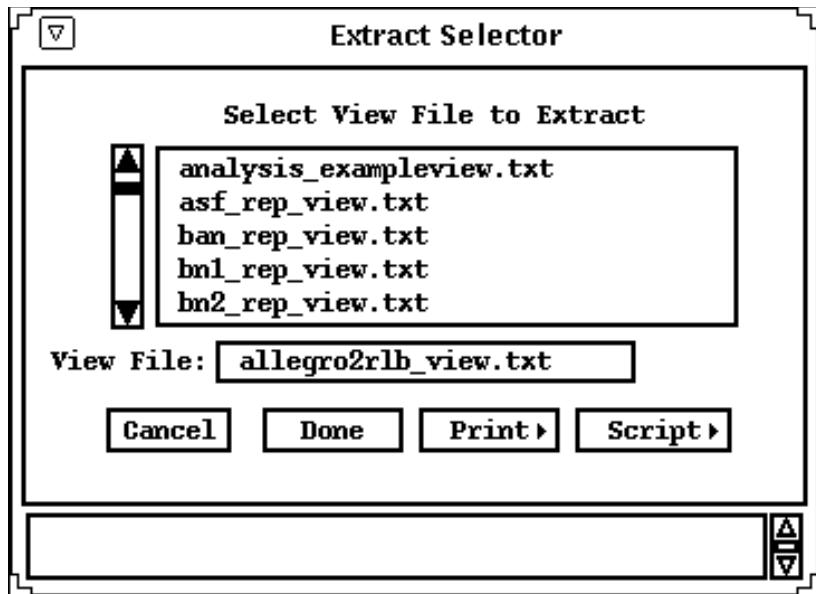
The command displays the **Extract Selector** form, listing all available extract view files.

3. Select any file in the list, or type a name into the *View File* field.

## Allegro SKILL Reference

### Form Interface Functions

allegro2rlb\_view.txt is entered.



```
axlFormGetField( form "view_file")
⇒ "allegro2rlb_view.txt"
```

Examines the value of "*view\_file*".

## **axlFormGetFieldRange**

```
axlFormGetFieldRange (
  r_form
  t_field
)
⇒ l_fieldRange/nil
```

### **Description**

Returns the minimum/maximum values specified for an INTFILLIN or REALFILLIN. For other field types, or if these values have not been set on the field, nil is returned.

### **Arguments**

<i>form</i>	Form dbid.
<i>t_field</i>	Name of the field.

### **Value Returned**

l_fieldRange/nil	nil, if field doesn't exist
	nil, if field isn't INTFILLIN or REALFILLIN
	nil, if the field doesn't have range set
list (min max)	If field has legal range set

## **axlFormGetValue**

```
axlFormGetValue(  
    t_formName  
    t_fieldName  
)  
⇒ field value/nil
```

### **Description**

Retrieve the value of the field in the specified form.

If the form is not open, the field doesn't exist, or any other error, `nil` is returned.

**Note:** This function will not do any mapping for you. If Cadence changes a form or field name, you must update your code to the new values.

### **Arguments**

<i>t_formName</i>	Name of form to query (see <code>jrl</code> contents for names)
<i>t_fieldName</i>	Name of field on form (see <code>jrl</code> contents for name)

### **Value Returned**

`value` or `nil`

## **axlFormGridGetSize**

```
axlFormGridGetSize(  
    r_form  
    t_field  
)  
⇒ list(rows columns)/nil
```

### **Description**

Returns the size of the grid as a list of two elements, the rows and the columns, if grid is initialized. If grid has not been initialized, or field is not a grid, returns nil to indicate error.

### **Arguments**

<i>r_form</i>	Standard form handle
<i>t_field</i>	Standard field name

### **Value Returned**

list (rows columns)	If successful
nil	On error

## **axlFormListGetCount**

```
axlFormListGetCount(  
    r_form  
    t_field  
)  
⇒ x_count/nil
```

### **Description**

Returns a count of number of items in the specified list box.

### **Arguments**

<i>r_form</i>	Form control
<i>t_field</i>	Name of the field

### **Value Returned**

<i>x_count</i>	Number of items in the list
<i>nil</i>	If not a list box

### **See Also**

[axlFormListGetSelCount](#)

### **Example**

See `axlform.il` example.

## **axlFormGridSelected**

```
axlFormGridSelected(  
    r_form  
    t_field  
) -> lx_selected/nil
```

### **Description**

This returns the selected item in a multi-select grid control. This should only be used if grid is running with the multi-select row option.

### **Arguments**

r_form	standard form handle
t_field	standard field name

### **Value Returned**

Returns list of selected items in a multi-select grid or nil if not the correct control.

### **See Also**

[axlFormGridNewCell](#)

### **Examples**

See fgrid.il in <CDSROOT>/share pcb/examples/skill/form/grid

Pseudo code:

```
axlFormGridEvents(fg "grid" 'mrowselect)  
;; select items  
selected = axlFormGridSelected(fg "grid")  
; if form select rows 5,6,7 (click on 5, then Shift click on 7)  
;; select items  
selected = axlFormGridSelected(fg "grid")  
-> (5 6 7)
```

## **axlFormGridSelectedCnt**

```
axlFormGridSelectedCnt (
    r_form
    t_field
) -> x_cnt/nil
```

### **Description**

This returns the count of rows selected in a multi-select grid control. This should only be used if grid is running with the multi-select row option.

### **Argument**

<i>r_form</i>	standard form handle
<i>t_field</i>	standard field name

### **Value Returned**

Returns count of selected items or `nil` if wrong type of control

### **See Also**

[axlFormGridNewCell](#)

### **Examples**

See `fgrid.il` in `<CDSROOT>/share pcb/examples/skill/form/grid`

Pseudo code:

```
axlFormGridEvents(fg "grid" 'mrowselect)
; if form select rows all rows (Ctrl-A in grid)
;; select items
selected = axlFormGridSelectedCnt(fg "grid")
-> 16
```

## **axlFormGridSetSelectRows**

```
axlFormGridSetSelectRows (
    r_form
    t_field
    x_min
    x_max
    g_option
) -> x_cnt/nil
```

### **Description**

This allows setting, clearing or toggling of selection state for a grid in multi-select row mode.

### **Arguments**

<i>r_form</i>	standard form handle
<i>t_field</i>	standard field name
<i>x_min</i>	min row number
<i>x_max</i>	max row number
<i>g_option</i>	what to do
	t – set row as selected
	nil – clear row as selected
	'toggle – toggle selected state of row

### **Value Returned**

*t* if succeeded, *nil* if not a grid field or not in multi-row select mode

### **See Also**

[axlFormGridNewCell](#)

### **Examples**

See *fgrid.il* in <*CDSROOT*>/share pcb/examples/skill/form/grid

## Allegro SKILL Reference

### Form Interface Functions

---

Pseudo code:

```
axlFormGridEvents(fg "grid" 'mrowselect)  
■ set row 4 as selected  
  axlFormGridSetSelectRows(fg "grid" 4 4 t)  
■ clear rows 4 thru 8 being selected  
  axlFormGridSetSelectRows(fg "grid" 4 8 t)  
■ clear all rows  
  axlFormGridSetSelectRows(fg "grid" -1 -1 nil)  
■ toggle state of row 1  
  axlFormGridSetSelectRows(fg "grid" 1 1 'toggle)
```

## **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**

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of field.

### **Value Returned**

<i>t</i>	All items deleted properly.
<i>nil</i>	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 840.
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 840.

3. On the SKILL command line, enter:

```
axlFormSetField(form "file_list" "fu")
```

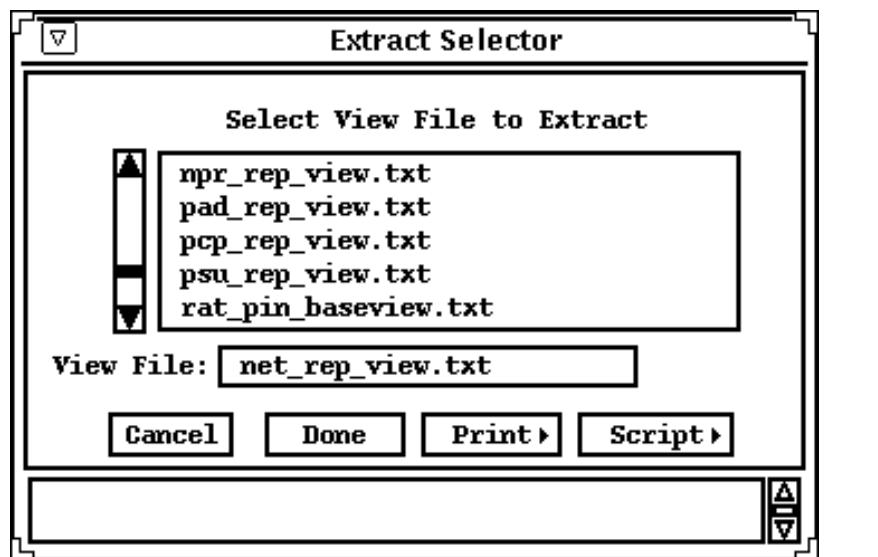
## Allegro SKILL Reference

### Form Interface Functions

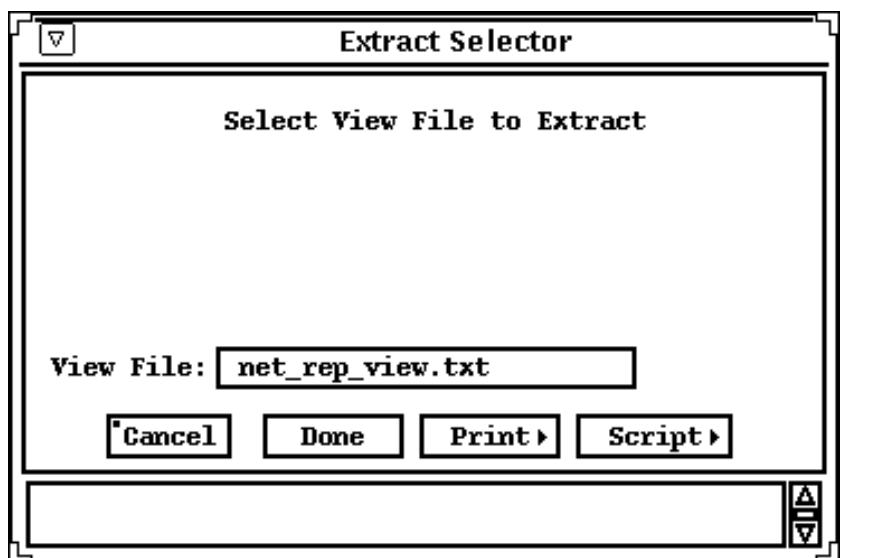
```
axlFormSetField(form "file_list" "bar")
axlFormSetField(form "file_list" nil)
==> t
```

The Extract Selector dialog box is displayed with new list as shown in [Figure 11-3](#) on page 841.

**Figure 11-1 Extract Selector Dialog Box**



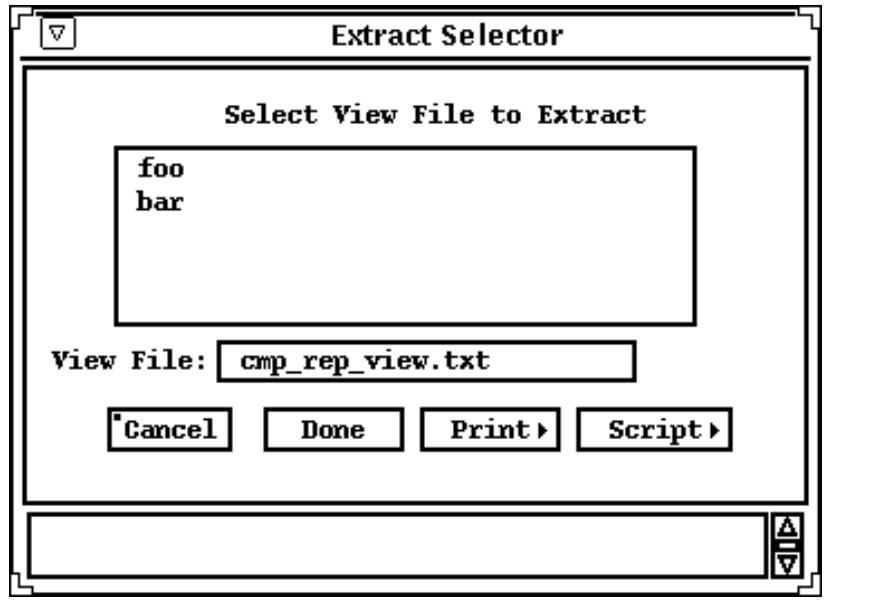
**Figure 11-2 Extract Selector Dialog Box - List removed**



## Allegro 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 Allegro 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**

<code>r_form</code>	Form id
<code>t_field</code>	Name of field.
<code>t_listItem/nil</code>	String of item in the list. Send <code>nil</code> to deselect any selected item and set list back to top.

### **Value Returned**

<code>t</code>	Highlights item. Arguments are valid.
<code>nil</code>	Arguments are invalid.

## **axlFormSetEventAction**

```
axlFormSetEventAction(  
    r_form  
    g_callback  
) -> t/nil
```

### **Description**

This function allows the user to register a callback function to be called whenever the user changes to a new active cell in the form. The callback registered during `axlFormCreate` dispatches events only when the user modifies a field value on the form (on exit from the field). This function allows the caller to receive an event when a field is first entered.

### **Arguments**

<i>r_form</i>	Form <i>dbid</i>
<i>g_callback</i>	Specifies the SKILL command(s) (callback(s)) to be executed whenever a new field is activated. The setting can be one of two formats:  <i>t_callback</i> : the string representation of the SKILL command(s) to be executed  <i>s_callback</i> : the symbol of the SKILL function to be called (the function is passed the <i>r_form</i> returned from <code>axlFormCreate</code> as its only parameter).

### **See Also**

[axlFormBNFDoc](#) and [axlFormCreate](#)

### **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**

```
form = axlFormCreate( MyForm  
    "extract_selector.form"  ' ("E"  "OUTER")
```

## **Allegro SKILL Reference**

### Form Interface Functions

---

```
'_formAction t)
axlFormSetEventAction( form '_formEventAction)
```

## **axlFormSetField**

```
axlFormSetField(  
    r_form  
    t_field  
    g_value/nil  
)  
⇒ t/nil
```

### **Description**

Sets values of individual form fields. These values may differ and depend on type of the field. For a complete discussion of field types, see the discussion at the front of this section.

Special notes for certain controls:

#### ■ LIST TYPE

Value may be a string, integer or real. Items are converted to strings before being displayed. A `nil` is needed to display the list.

Alternatively, value may be a list of strings. This results in better performance when you have many items to display.

#### ■ COLOR TYPE

`g_value` parameter may have several types:

<code>s_colorSymbol</code>	Set field to predefined color
<code>x_number</code>	Set field to product color
<code>t</code> or <code>nil</code>	Depress or raise field
<code>l_both</code>	A list allows setting both check and value; pass a list of the color set

`s_colorSymbol` May be black, white, red, green, yellow, multivalue, button or default.

`x_number` is an integer between 1 and 24 with 0 being background.

#### ■ CHECKBOX

The values that unset the checkbox are: `nil`, `0`, `"nil"`, `"false"` and `"no"`. All other values set the checkbox.

#### ■ TRACKBAR

If the field is a trackbar, two modes are supported.

- ❑ **g\_value = t**  
Moves the slider to the next position.
- ❑ **g\_value = integer**  
Absolutely sets trackbar to the indicated position.

## Arguments

*r\_form*                    Form *dbid*.  
*t\_field*                Name of field. Field name is a string or symbol.  
*g\_value*                Desired value of field. May be a string, boolean, integer or floating point number or a list; function of field type.

## Value Returned

*t*                        If Successful  
nil                       Failed

## Examples

### List Field (field is named "list")

```
; display 3 items in list
axlFormSetField(fw, "list", "a")
axlFormSetField(fw, "list", "b")
axlFormSetField(fw, "list", "c")
; nil required first time list is displayed
axlFormSetField(fw, "list", nil)

; display 3 items in list - alternative
axlFormSetField(fw, "list", '("a" "b" "c")')
```

### Color field (field is named "color")

```
; sets the color field to pre-defined color "red"
axlFormSetField(fw, "color", `red)
; sets the color field to product color 1
axlFormSetField(fw, "color", 1)
```

## **Allegro SKILL Reference**

### Form Interface Functions

---

```
;; visually depresses the color field if not greyed
axlFormSetField(fw, "color", t)
;; visually depresses the color field and set to
;;      pre-defined green color
axlFormSetField(fw, "color", '(green t))
```

#### **Tab field (field is named "tab")**

```
;; puts the tab on top
axlformSetField(fw, "tab", nil)
```

## **axlFormSetFieldHelpTip**

```
axlFormSetFieldHelpTip(  
    r_form  
    t_field  
    t_value  
)  
⇒ t/nil
```

### **Description**

Sets the help tip associated with the indicated field. The value may be either a simple string or an HTML with a proper structure. Note that HTML is only recommended when the TOOLTIPHELP tag is set on your form to enable mouse-over help instead of INFO field text to display field help.

### **Arguments**

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Name of field.
<i>t_value</i>	Help tip string to associate with the field, <i>nil</i> to clear existing

### **Value Returned**

<i>t</i>	Returns <i>t</i> if the help tip was set to desired value.
<i>nil</i>	Otherwise

### **Examples**

```
axlFormSetFieldHelpTip(form "my_field" "This field is for entering data")
```

See the use of `axlFormSetField` in `axlFormCreate` examples.

## 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 789.

```
axlFormSetInfo( form "file_list" fileName)
```

## **axlFormSetMouseActive**

```
axlFormSetMouseActive(  
    r_form  
)  
==> t/nil
```

### **Description**

Sets the option to dispatch the MouseActive event on a form.

While this can be used to display dynamic help on a per field basis (this is what the example code does) a better method exists called the "helptip" which is driven from the form file. See the axlFormBNFDoc (note 12).

### **Arguments**

*r\_form*                          Handle for the form

### **Value Returned**

*t*                                  Option was set

*nil*                                *r\_form* does not reference a valid form

### **Example**

See <cdsroot>/share pcb/examples/form/basic

## **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 Allegro 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 789 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**

<i>t</i>	Changed form title.
<i>nil</i>	No form title changed.

### **Example**

See “[AXL Forms: Example 1](#)” on page 789.

```
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	<i>r_form</i> is the <i>dbid</i> of a form.
nil	<i>r_form</i> is not the <i>dbid</i> 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 Allegro 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 11-4 on page 859](#) and [Table 11-5 on page 861](#) show the available field types and how they impact the *r\_form* data type.

[Table 11-4 on page 859](#) describes Form Field Types using the following:

Type	What the user calls the field
Keyword	What the form file calls the field

## Allegro 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 11-4 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 <sup>1</sup>
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 <a href="#">Using Grids</a> on page 781	

<sup>1</sup>. 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 check boxes 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 11-5 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 <a href="#">Using Grids</a> on page 781
<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 Allegro 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 dependent on the field type. See [Table 11-4](#) on page 859 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, grayed and hidden fields.

## Arguments

*r\_form*                  Form *dbid*.

## Value Returned

*t*                  Always returns *t*.

## Example

See [axlFormCreate](#) on page 819 and [axlFormBuildPopup](#) on page 825 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 Allegro PCB Editor Colors with AXL-SKILL](#) on page 77 for the allowed values.

Other form controls support color as a fundamental part of their interface. These are *COLOR* (See [Accessing Allegro PCB Editor Colors with AXL-SKILL](#) on page 77) and *GRID* (See [Using Grids](#) on page 781) controls.

### **Restrictions:**

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.



*Caution*

***If setting the background of a ENUM field on Windows, you must set the OPTION color in the form file as in the following example.***

```
FIELD enum_control
FLOC 16 12
ENUMSET 41
OPTIONS color
POP "enum_popup"
ENDFIELD
```

***This is the same option as the 'ownerdrawn'. The color option is ignored on UNIX which does not support themes. For theming purposes, Microsoft takes the background color so setting this option disables Microsoft themes for this control.***

## 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 axlColorDoc) or nil

## Value Returned

<i>t</i>	Color changed.
nil	Error due to an incorrect argument.

## Example 1

You can find an example in axlform.il.

## **Allegro SKILL Reference**

### Form Interface Functions

---

```
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(f1s "string" 'text nil)  
axlFormColorize(f1s "string" 'background nil)
```

Sets control back to default.

#### **Example 4**

```
axlFormColorize(f1s "string" 'background 1)
```

Sets control background to Allegro 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**

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.

### **Value Returned**

<i>t</i>	Success.
<i>nil</i>	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=>,n>*, *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 <code>cellselect</code> .
'mrowselect	Puts grid into multi-row select mode. This is mutually exclusive with <code>cellselect</code> . Use <a href="#">axlFormGridSelected</a> to determine what rows are selected.
'cellselect	Puts grid into cell select mode. This is mutually exclusive with <code>rowselect</code> and <code>mrowselect</code> .
'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 ' <code>rightpopup</code> 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.

## Allegro SKILL Reference

### Form Interface Functions

---

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 871 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 781 for a grid overview.

Assigning events to a grid overrides the previous assignment. Therefore, following do not work:

```
axlFormGridEvents(fw "grid 'change")
axlFormGridEvents(fw "grid 'cellselect")
```

Instead, use the following command.

```
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.

## See Also

[axlFormGridNewCell](#)

## 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 781 for an overview.

[Table 11-6](#) on page 876 describes the `FormGridCol` attributes.

**Table 11-6 FormGridCol Attributes**

Attribute	Type	Default	Description
fieldType	symbol	TEXT	Field types include: TEXT, STRING, LONG, REAL, ENUMSET, and CHECKITEM.
fieldLength	integer	16	Maximum data length.
colWidth	integer	0	Width of column.
headText	n/a	n/a	If the grid has a top heading, sets the heading text. Can also set using <code>axlFormGridSet</code> .

Alignment Types (left, right, and center):

align	symbol	Left	Column alignment.
topAlign	symbol	Center	Column header alignment.

## Allegro SKILL Reference

### Form Interface Functions

---

**Table 11-6 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 white space or the symbol '!'.
helpTip	string	n/a	Text to display when user hovers over the column header
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.

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:** 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

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.
<i>r_formGridCol</i>	Instance of type <code>formGridCol</code> .

## Value Returned

<i>t</i>	Column added.
<i>nil</i>	Failure due to a nonexistent form or field, field not of type GRID, errors in the <i>g_options</i> defstruct, or grid already had a row added.

## Examples

For a complete grid programming example, see: `<cdsroot>/share pcb/examples/skill/form/grid`.

### 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 [axlFormGridDoc](#) for grid overview.

See [axlFormGridSetBatch](#) on page 883 for a complete description of cell attributes.

### **Arguments**

None.

### **Value Returned**

<i>r_cell</i>	New list gridcell handle.
---------------	---------------------------

### **See Also**

[axlIsGridCellType](#), [axlFormGridInsertRows](#), [axlFormGridDeleteRows](#),  
[axlFormGridCancelPopup](#), [axlFormGridEvents](#), [axlFormGridOptions](#), [axlFormGridSetBatch](#),  
[axlFormGridBatch](#), [axlFormGridGetCell](#), [axlFormGridReset](#), [axlFormGridSelected](#),  
[axlFormGridSelectedCnt](#), [axlFormGridSetSelectRows](#)

## **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**

<i>r_form</i>	Form handle.
<i>t_field</i>	Field name.

### **Value Returned**

<i>t</i>	Grid reset.
<i>nil</i>	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, `fgrid.il` at `<cdsroot>/share pcb/examples/skill/form/grid`. 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>s_backColor</code>	Optional background color.
<code>s_textColor</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.
<code>s_invisible</code>	Make cell invisible. Current cell settings are preserved by the grid.

## Allegro SKILL Reference

### Form Interface Functions

---

<i>s_popup</i>	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.
<i>t_objType</i>	Object name " <i>r_cell</i> " (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:

- |                      |                                 |
|----------------------|---------------------------------|
| (< <i>row</i> >, 0)  | Set side header display text.   |
| (< <i>row</i> >, -1) | Set side header scripting text. |

**Note:** Case is ignored, and text must not contain spaces or the '!'.

- |                     |  |
|---------------------|--|
| (0, < <i>col</i> >) | Set top header display text. You may also set the top header at column creation time using <code>axlFormGridInsertCol</code> . |
| (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

<code>r_form</code>	Form handle.
<code>t_field</code>	Field name.
<code>t_callback</code>	Function to callback. Takes a single argument: <code>g_pvtData</code>
<code>g_pvtData</code>	Private data (Pass <code>nil</code> if not applicable.)

## Value Returned

<code>t</code>	Grid cell changed.
<code>nil</code>	No grid cell changed, or application callback returned <code>nil</code> .

## **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**

<i>r_form</i>	Form handle.
<i>t_field</i>	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 grayed, 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 <i>nil</i> 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.

## **Allegro SKILL Reference**

### Form Interface Functions

---

#### **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 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.
<i>nil</i>	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**

*r\_form* Form control.

*t\_field* Name of the field.

### **Values Returned**

*nil* If not a multi-list box.

*x\_count* Number of items selected.

### **See Also**

[axlFormListGetSelItems](#), [axlFormListGetCount](#)

### **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 <code>nil</code> 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**

<code>r_form</code>	Form id.
<code>t_field</code>	Field name.
<code>s_option</code>	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 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 767.) 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 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")
```

## **axlFormGetFieldRange**

```
axlFormGetFieldRange (
  r_form
  t_field
)
⇒ l_fieldRange/nil
```

### **Description**

Returns the min/max values specified for an INTFILLIN or REALFILLIN value. For other field types, or if these values are not set on the field, `nil` is returned.

### **Arguments**

<code>r_form</code>	Form <i>dbid</i> .
<code>t_field</code>	Name of the field.

### **Value Returned**

<code>l_fieldRange/nil</code>	<code>nil</code> if field doesn't exist
	<code>nil</code> if field isn't INTFILLIN or REALFILLIN
	<code>nil</code> if the field doesn't have range set
	<code>list(min max)</code> if field has legal range set

## **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 858 for a list of supported field types.

### **Arguments**

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Field name.

### **Value Returned**

<i>g_fieldType</i>	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>abid</i> .
<i>t_field</i>	Field name to establish as new button default.
<i>g_mode</i>	<code>t</code> to enable the default button in the form, <code>nil</code> to disable it.

### **Value Returned**

<code>t</code>	Default button set.
<code>nil</code>	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 781 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 axlFormGridEvents 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**

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Field name.

### **Value Returned**

<i>t</i>	Field set active.
<i>nil</i>	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 nil for all fields.
<i>x_decimalPlaces</i>	Number of decimal places - must be a positive integer.

### **Value Returned**

t	Successfully set new decimal precision.
nil	Error due to invalid arguments.

## **axlFormSetFieldEditable**

```
axlFormSetFieldEditable(  
    r_form  
    t_field  
    g_editable  
)  
⇒ t/nil
```

### **Description**

Sets individual form fields to editable (t) or grayed (nil).

### **Arguments**

<i>r_form</i>	Form <i>dbid</i> .
<i>t_field</i>	Field name.
<i>g_editable</i>	Editable (t) or grayed (nil).

### **Value Returned**

t	Set form field to editable or grayed.
nil	Failed to set form field to editable or grayed.

## **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.

TRACKBAR: When used for a trackbar field type. If either of the argument is 0, then the current setting for that option is maintained. Both *g\_min* and *g\_max* must be integer numbers. *g\_min* is the interval of the tickmarks (default is 1). Tickmarks may not be shown in all user interfaces. This is the interval size when moving the trackbar to the next tick mark via axlFormSetField. *g\_max* is the number of steps in the trackbar (default is 100).

### **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, indicating a problem with one of the input parameters.

## 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*, the item is added as the last child of a parent. If parent is *nil*, item is created as the root of the tree.

**Note:** This is the only interface for adding 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.

The 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, 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*.

## Callback Values

In the callback function for the form, the first argument *form*, has the following properties relevant to treeviews:

<i>form-&gt;curValue</i>	Contains the label of a treeview item. This is set in single select mode and in multi select mode when the user selects the item. In this case, the <i>result-&gt;tree.selectState</i> is -1.
<i>form-&gt;curValueInt</i>	Contains id of the selected treeview item.
<i>form-&gt;selectState</i>	In multi select mode, when the user picks the selection checkbox, this field will contain:  0 if selection checkbox is not checked  1 if selection checkbox is checked  2 if selection checkbox is disabled ((3 state mode only)  In this case the <i>result-&gt;string</i> is empty.  In all other cases the value is -1.
<i>form-&gt;event</i>	If event property is set to "rightpopup", treeview control has a popup and the user has selected an item in the popup. In this case, <i>form-&gt;curValue</i> is set to the popup index selected, and <i>form-&gt;selectState</i> is set to -1. <i>form-&gt;curValueInt</i> 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."

## 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 the parent. If <i>null</i> , item created as the root of the tree.
<i>g_hInsertAfter</i>	Handle of the sibling to add the item after. If <i>null</i> , item added at the end of siblings of the parent.
<i>t_multiSelectF</i>	If <i>t</i> , the item has a checkbox for multi selection.
<i>g_hLeafImage</i>	Handle of the image to use whenever this item is a leaf node in the tree view. If <i>nil</i> or not supplied, the default pink diamond image is used.
<i>g_hOpenImage</i>	Handle of image to be used whenever this item is an expanded parent node in the tree view. If <i>nil</i> or not supplied, the default open folder image is used.
<i>g_hClosedImage</i>	Handle of image to use whenever the item is an unexpanded parent node in the tree view. If <i>nil</i> or not supplied, the default closed folder image is used.

## Value Returned

<i>g_hItem</i>	Item is added to the tree view control.
<i>nil</i>	No item is added to the tree view control due to an error.

## Example

see <cdsroot>/share pcb/examples/skill/form/basic/axlform.il

## **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.

**Value Returned**

- |     |   |
|-----|---|
| t   | Tree view item's images are modified.     |
| nil | Failed to modify tree view item's images. |

**See Also**

[axlFormTreeViewLoadBitmaps](#)

## **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**

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**

<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>t_label</i>	Text of given tree view item.
<i>nil</i>	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 of a treeview item. 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 Allegro 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 is found using BMPPATH variable.
- Uses file extension .bmp
- Bitmap images must be 16 x 16 pixels.
- A bitmap file can contain more than one image provided they are appended horizontally (i.e. a bitmap file containing n images will be (16\*n) x 16 pixels).
- The 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. The list is ordered to correspond with the order that the images were listed in the <i>lt_bitmaps</i> parameter.
<code>nil</code>	One or more of the bitmap files could not be found, or an error was encountered while adding images.

## See Also

[axlFormTreeViewChangeImages](#), [axlFormTreeViewAddItem](#)

## Example

- File `myBmp1.bmp` is a 16 x 16 bitmap and `myBmp2.bmp` is a 32 x 16 bitmap:

```
tree = axlFormTreeViewAddItem(fw "tree" "one" nil nil 'TVSELECT_3STATE)
l = axlFormTreeViewLoadBitmaps(fw "tree" list("myBmp1" "myBmp2"))
axlFormTreeViewChangeImages(fw "tree" tree car(l) cadr(l) nil)
```

`axlFormTreeViewLoadBitmaps` may return (6 7 8) into variable `l`.

This mean that the image handle 6 would refer to the bitmap contained in `myBmp1.bmp`, the image handle 7 would refer to the left half of `myBmp2.bmp`, and the image handle 8 would refer 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

<i>r_form</i>	Form id.
<i>t_field</i>	Field name as a string
<i>s_option</i>	<i>s_option</i> is one of the symbols listed.
<i>g_hItem</i>	Tree view user data type.
[ <i>s_data</i> ]	<i>optional symbol</i>

Each *s\_option* 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*. The pairs of different values of *s\_option* with *g\_hItem* is in the following manner.

<i>s_option</i>	<i>g_hItem</i>
TV_REMOVEALL	<i>nil</i>
TV_DELETEITEM	<i>g_hItem</i>
TV_ENSUREVISIBLE	<i>g_hItem</i>
TV_EXPAND	<i>g_hItem (4)</i>
TV_EXPAND_TOP	<i>nil (5)</i>
TV_COLLAPSE	<i>g_hItem (4)</i>
TV_COLLAPSE_TOP	<i>nil (5)</i>
TV_SELECTITEM	<i>g_hItem (2) (3)</i>

## Allegro SKILL Reference

### Form Interface Functions

---

TV_SORTCHILDREN	<i>g_hItem</i>
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	<i>g_hItem</i> (1)
TV_NOSHOWSELALWAYS	nil
TV_SHOWSELALWAYS	nil

#### Notes:

- 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* is the Handle of item in the tree view control. Its value can be received 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.
- If `nil` is passed for TV\_EXPAND or TV\_COLLAPSE then all levels (including) children are expanded or collapse.
- The two \_TOP options, TV\_EXPAND\_TOP and TV\_COLLAPSE\_TOP, respectively, expand and collapse all top level tree items. Children tree item states are preserved.

### Value Returned

t	Changed one or more items in tree view control.
nil	Failed to change items in tree view control.

### Example

- Delete selected treeview item in a form.

```
(axlFormTreeViewSet form form->curField 'TV_DELETEITEM form->curValue)
```

- Expand all levels including children:

```
axlFormTreeViewSet(form "tree" 'TV_EXPAND nil)
```

- Collapses all expanded top levels:

```
axlFormTreeViewSet(form "tree" 'TV_COLLAPSE_TOP nil)
```

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"><li>■ Select state is unchecked if <i>g_state</i> is nil or 'TVSTATE_UNCHECKED</li><li>■ Select state is checked if <i>g_state</i> is t or 'TVSTATE_CHECKED</li><li>■ Select state is disabled if <i>g_state</i> is 'TVSTATE_DISABLED</li></ul>

### **Value Returned**

t	Set select state.
nil	Failed to set select state.

## **Allegro SKILL Reference**

### Form Interface Functions

---

**Allegro SKILL Reference**  
Form Interface Functions

---

---

## **Simple Graphics Drawing Functions**

---

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. (For more information on thumbnails, see the axl form BNF document, look for the entries with THUMBNAIL).

`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.

## Allegro SKILL Reference

### Simple Graphics Drawing Functions

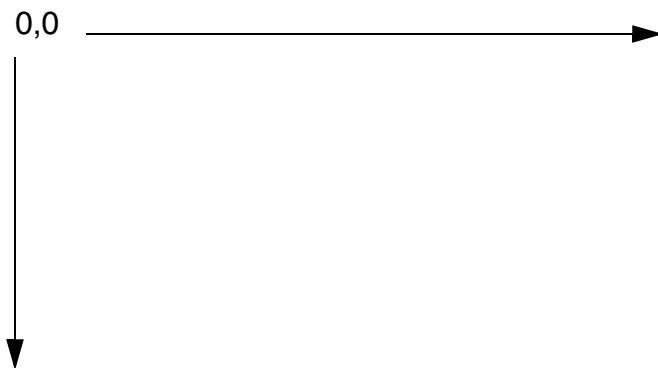
---

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.

## Allegro SKILL Reference

### Simple Graphics Drawing Functions

---

#### 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	(represents the current button color) The color can be SVG color names provided by <a href="#">WWW consortium</a> . <ul style="list-style-type: none"><li>■ #RGB (each of R,G,B is a single hex digit)</li><li>■ #RRGGBB</li><li>■ #AARRGGBB</li><li>■ #RRRGGGBBBB</li><li>■ #RRRRGGGGBBBB</li></ul> Standard colors are: black, white, red, green, yellow, blue, light blue, rose, purple, teal, dark pink, dark magenta, aqua, gray, olive, orange, pink, beige, navy, violet, silver, rust, lime, brown, mauve, magenta, light pink, cyan, salmon, peach, dark gray, button
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

**Table 12-1 r\_graphics Option Properties, continued**

text_bkmode	transparent	Text background display mode. Applies only to text.	transparent, opaque
-------------	-------------	--	---------------------

The following geometric elements are not supported:

- Arcs
- Shapes (for example, polygon's with arcs)
- Bitmaps from resources
- Ellipses

Additionally, the following capabilities are not supported:

- Geometric transformations (rotation, translation, scaling)
- Transparency
- Loading bitmaps from the resource file
- No 3 or 2.5 D

## Examples

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 a form draw window (drawing area in the graphics field). More drawing can take place on top of the bitmap.

#### Arguments

<i>r_graphics</i>	Graphics handle.
<i>t_bitmap</i>	Name of bitmap file. File must be on the BMPPATH, with .bmp as the extension.

#### Value Returned

<i>t</i>	Bitmap loaded into drawing area in the graphics field.
<i>nil</i>	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**

Use this command to set graphics callback in a form field. It 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_Xextent  
    x_Yextent  
)  
⇒ t/nil
```

### **Description**

Forces a draw in a mapped window. 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_Xextent</i>	X-extent of the drawing window.
<i>x_Yextent</i>	Y-extent of the drawing window.

### **Value Returned**

t	Successful
nil	Error occurred due to invalid arguments.

### **See Also**

[axlGRPDoc](#)

## **axlGRPDrwPoly**

```
axlGRPDrwPoly(  
    r_graphics  
    l_vertices  
)  
⇒ t/nil
```

### **Description**

Draws a polygon (multi-segment 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 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**

Force call to register callback function for a draw window. 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 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*:

---

<b>Severity Level</b>	<b>Type</b>	<b>Type Description and Response by AXL Message Handler</b>
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 SKILL Reference

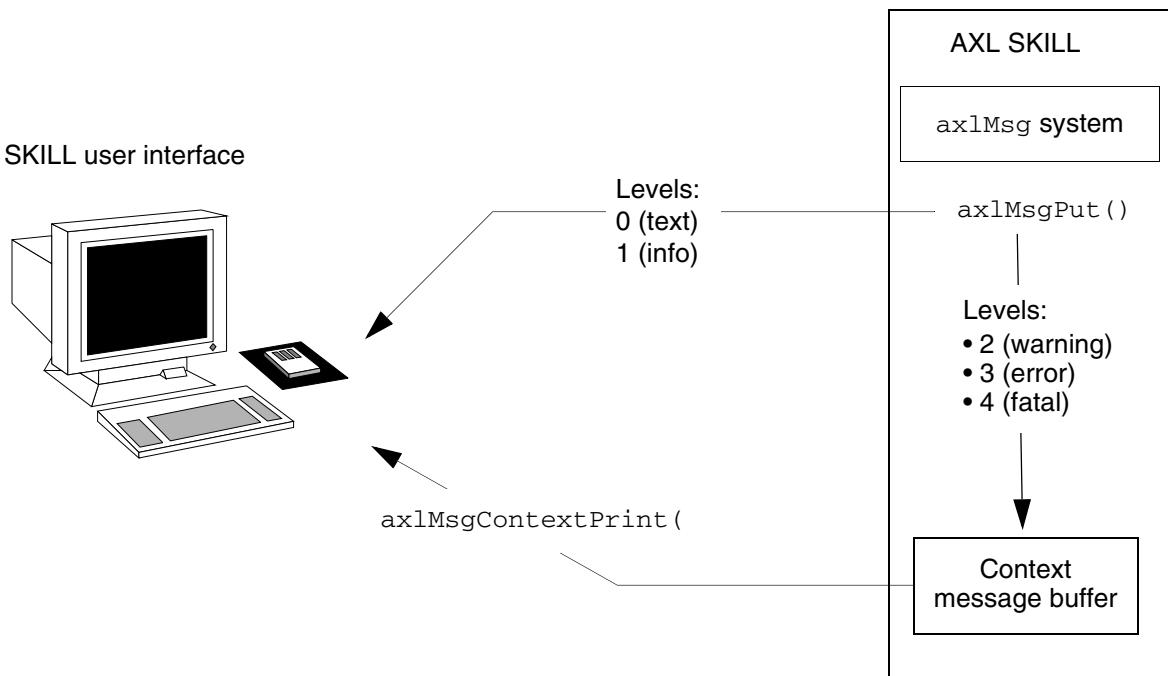
### Message Handler Functions

---

<b>Severity Level</b>	<b>Type</b>	<b>Type Description and Response by AXL Message Handler</b>
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

*g\_message\_format*      Context message (`printf` like) format string. See “[Overview](#)” on page 943 for a description of messages and valid arguments.

*g\_arg1 ...*      Values for substitution arguments for the format string.

#### Value Returned

*t*      Always returns *t*.

#### Example

See the “[Message Handler Example](#)” on page 945.

```
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 t.
---	-------------------

### **Example**

See the “[Message Handler Example](#)” on page 945 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
```

## **axIMsgContextGetString**

```
axIMsgContextGetString(  
    r_context  
)  
⇒ lt_messages/nil
```

### **Description**

Gets the messages in the message buffer and removes them from the buffer. Call `axIMsgContextGetString` subsequently to communicate those messages to the user (for example, in a log file).

### **Arguments**

<i>r_context</i>	Context handle from <code>axIMsgContextStart</code> . Gets messages only for this context (or any children). An argument of <code>nil</code> causes <code>axIMsgContextGetString</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 945.

```
axIMsgContextGetString(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**

<i>lt_format_strings</i>	List of format strings of the buffered messages.
<code>nil</code>	No buffered messages found.

### **Example**

See the “[Message Handler Example](#)” on page 945.

```
mylist = axlMsgContextGet(context)  
⇒ ("My warning" "My error" "My fatal error %s")
```

## **axlMsgContextTest**

```
axlMsgContextTest(  
    r_context  
)  
⇒ x_class
```

### **Description**

Returns the most severe message class of the messages in the context message buffer. See [axlMsgContextInBuf](#) on page 951 to check for a particular message class.

### **Arguments**

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . Looks only for messages for this context. If <i>r_context</i> is nil, <code>axlMsgContextTest</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 945.

```
printf("Message severity %d\n" axlMsgContextTest(context))  
⇒ Message severity 4
```

## axIMsgContextInBuf

```
axIMsgContextInBuf(  
    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 axIMsgContextStart. Looks only for messages in this context (or any children). If <i>r_context</i> is nil, axIMsgContextInBuf looks through all contexts.
<i>t_format_string</i>	Format string of the message.

### Value Returned

<i>t</i>	Specified message is in the buffer.
nil	Specified message is not in the buffer.

### Example

See the [“Message Handler Example”](#) on page 945.

```
if( axIMsgContextInBuf(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 axlMsgContextStart. Removes messages for only this context (or any children). If <i>r_context</i> is nil, axlMsgContextRemove 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 945.

```
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

<i>g_format_string</i>	Context message (printf like) message format string.
<i>g_arg1</i> ...	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

<i>r_context</i>	Context handle to use in subsequent axlMsgContext calls.
------------------	--

### Example

See the [“Message Handler Example”](#) on page 945.

```
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 945.

```
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**

<i>r_context</i>	Context handle from <code>axlMsgContextStart</code> . If <i>r_context</i> is nil, clears all messages in all contexts.
------------------	--

### **Value Returned**

<i>t</i>	Always returns <i>t</i> .
----------	---------------------------

### **Example**

See the “[Message Handler Example](#)” on page 945.

```
context = axlMsgContextStart()  
... do some things ...  
axlMsgContextClear(context)
```

## **axIMsgCancelPrint**

```
axIMsgCancelPrint()  
    ⇒ 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 945.

```
axIMsgCancelPrint()
```

Sets severity level to 2 (warning) and puts the following into the message buffer (if axIMsgSys 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 945.

```
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 945.

```
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 945.

```
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 945.

```
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 currently active layout. This is the drawing name in the main window's title bar.

### **Arguments**

None.

### **Value Returned**

*axlCurrentDesign*      Returns the current design name as a string. There is always a current name, even if it is "unnamed" (*unnamed.brd*).

### **See Also**

[axlGetDrawingName](#), [axlOpenDesign](#)

### **Example**

```
axlCurrentDesign () ⇒ "mem32meg"
```

Gets the name of the currently active layout.

## **axlDesignType**

```
axlDesignType(  
    g_option  
)  
⇒ t_type/lt_type
```

### **Description**

Returns the type of design as a string.

### **Arguments**

*g\_option*                    Possible values are:

nil        Return design type domain.

Examples are:

    LAYOUT for .brd or .mcm

    SYMBOL for .dra

*t*        fully qualified design type. Examples are:

    For layout: "board", "mcm", "mdd" (module)

    For symbol: "package", "mechanical", "format"  
        "shape" or "flash"

- 'domain — return universe of domains (strings may be repeated). This is all possible values that nil may return.
- 'members — return list of fundamental design types.
- 'symbols — return list of symbol types. If a "SYMBOL" this is sub-type of the dra.

The combination of 'members and 'symbols is the universe of the 't' option.

## Value Returned

<i>t_type</i>	string describing design type.
<i>lt_type</i>	list of strings in a universe.

## See Also

[axlIsSymbolEditor](#)

## Example

- Gets the high-level design type of the current active layout.

```
axlDesignType(nil)  
⇒ "LAYOUT"
```

- Gets the detailed design type.

```
axlDesignType(t)  
⇒ "BOARD"
```

## **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 <a href="#"><u>axlDesignType</u></a> ).

### **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 Allegro PCB Editor “*create symbol*” command and thus has the same behavior.

## **axlSetSymbolType**

```
axlSetSymbolType(  
    t_symbolType  
)  
⇒ t_symbolType/nil
```

### **Description**

Sets the Allegro PCB Editor symbol type. Has some minor effect on the commands available and the edit checks performed when the symbol is “compiled” ([axlCompileSymbol](#)).

Use [axlDesignType](#) to determine current symbol type.

### **Arguments**

<i>t_symbolType</i>	"package", "mechanical", "format", "shape" or "flash"
---------------------	--

### **Value Returned**

<i>t_symbolType</i>	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*.

### **See Also**

[axlCompileSymbol](#), [axlDesignType](#)

## axIDBControl

```
axIDBControl(  
    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 displays the current setting, the setting may not be updated. Additional side effects of individual controls are listed.

**Note:** Several display options have been moved to axIDBDisplayControl but for backward compatibility are still supported using this interface.

Items currently supported for *s\_name* include the following:

Name: *busRats*

Value: t or nil

Set?: Yes

Description: Queries and changes the bus rats option

Equivalent: Prmed Display group

Side Effects: If changing should do ratsnestDistance first

Symphony Supported: Yes

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.

Symphony Supported: Yes

Name: *fixedPattern*

Value: nil (disabled), 1 max pattern index (enabled)

Set?: Yes

Description: Sets the pattern index for fixed objects.

Equivalent: Same as color192 form Display tab fixed pattern field.

Side Effects: Display updates.

Symphony Supported: Yes

## Allegro SKILL Reference

### Design Control Functions

---

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.

Symphony Supported: Yes

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

Symphony Supported: Yes

Name: *maxTextBlock*

Value: 16 to 64

Set?: No

Description: Reports the maximum text block number.

Equivalent: None

Side Effects: None

Symphony Supported: Yes

Name: *activeLayer*

Value: <class> / <subclass>

Set?: Yes

Description: Accesses the current class/subclass. To obtain subclass do

```
cadrl(parseString(axlDBControl('activeLayer "/')) )
```

Equivalent: Same as ministatus display.

Side Effects: None

Symphony Supported: Yes

Name: *activeAltLayer*

Value: <subclass>

Set?: Yes

Description: Accesses the current alternative etch layer

Equivalent: Same as ministatus display when in add connect.

Side Effects: None

Symphony Supported: Yes

## Allegro SKILL Reference

### Design Control Functions

---

Name: *defaultSymbolHeight*

Value: float

Set?: Yes

Description: Sets the default symbol height in the database.

Equivalent: Prmed from DRC Symbol Height.

Side Effects: Will set DRC out of date, and does not update status dialog box if it is present.

Symphony Supported: No

Name: *editingTime*

Value: int

Set?: Reset

Description: Reports editing time of design in minutes.

Equivalent: status from

Side Effects: Can reset time by passing `g_value == t`

Symphony Supported: Yes

Name: *symbolRotation*

Value: float

Set?: Yes

Description: Sets the initial symbol rotation

Equivalent: Prmed form's symbol angle

Side Effects: None

Symphony Supported: No

Name: *symbolMirror*

Value: t/nil

Set?: Yes

Description: Sets the initial symbol mirror.

Equivalent: Prmed form's Symbol mirror.

Side Effects: None

Symphony Supported: No

Name: *schematicBrand*

Value: none, concepthdl, capture

Set?: Yes

Description: Queries current database schematic branding. You can set the branding only to 'none' via SKILL.

Equivalent: See netin in the Allegro PCB Editor dialog.

Side Effects: None.

Symphony Supported: No

Name: *cmgrEnabledFlow*

Value: t or nil

Set?: Yes (only if flag is set)

## Allegro SKILL Reference

### Design Control Functions

---

Description: Reports if board is in constraint manager enabled flow. Only valid in HDL Concept Flow.  
t - Cadence schematic Constraint Manager 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.

Symphony Supported: No

Name: *cdsxFil*e

Value: t or nil

Set?: t or nil

Description: Reports if a single .cdsxFil file was used by netrev instead of multiple .pst files.  
If t, then genfeed format will output a .cdsxFil file.

Symphony Supported: No

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

Symphony Supported: No

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

Symphony Supported: No

Name: *ecsetApplyRipupEtch*

Value: t or nil

Set?: Yes

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 "expert" level tools. Applies setting at system level to all open designs.

t - ripup etch  
nil - preserve etch

Equivalent: Same as Constraint Manager Tools — Options — “Rip up etch...”

Side Effects: None

Symphony Supported: No

Name: *maxEtchLayers*

Value: number

Set?: No

Description: Returns maximum number of etch subclasses.

Equivalent: none

Side Effects: none

Symphony Supported: Yes

Name: *dynamicFillMode*

Value: 'wysiwyg, 'rough, nil (Disabled) (must be a symbol not a string)

Set?: Yes

Description: Controls filling of dynamic shapes.

Equivalent: shape global param

Side Effects: Disabled in symbol editor.

Symphony Supported: No

Name: *maxLength*

Value: number

Set?: Yes

Description: Sets maximum name length. Minimum is 31 and maximum is 255. For designs, you cannot set it lower than the current length. For new designs, the initial value is set by the value in the env variable *allegro\_long\_name\_size*.

Equiv: Design Parameter Editor, Long Name Size

Symphony Supported: No

Name: *dynamicFilletsOn*

Value: t nil

Set?: Yes

Description: Controls filling of dynamic shapes

Equiv: Gloss param fillet - dynamic fillets option

Side Effects: Disabled in symbol editor and lower level PCB tools. When enabled will update design with fillets.

Symphony Supported: No

Name: *newFlashMode*

Value: t/nil

Set?: Yes

## **Allegro SKILL Reference**

### Design Control Functions

---

Description: Returns if board is running old style (nil) or new style (t) flash mode (WYSIWYG negative artwork using fsm files) Note WYSIWYG is now Smooth in the display.

Equiv: none

Side Effects: If you change the mode to t without updating flash symbols, wrong artwork may occur. You should only change the mode at design creation time before any padstacks with flashes are loaded into the design.

Symphony Supported: No

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.

Symphony Supported: Yes

Name: *dbSize*

Value: int

Set?: No

Description: returns current database size (memory footprint)

Equiv:none

Side Effects: none

Symphony Supported: Yes

Name: *retainElecCnsOnNets*

Value: t or nil

Set?: Yes

Description: Queries and changes the retain electrical constraints on nets

Equiv: netrev based option

Side Effects: This should only be set on new designs. With populated designs existing Electrical constraints will remain at xnet level. Setting the option will not promote existing settings to the xnet.

Symphony Supported: No

Name: *mirrorUserMask*

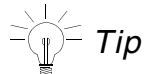
Value: t or nil

Set?: t or nil

Description: Most designs with padstack user mask layers offer Allegro mirror capability (layers with suffix \_TOP will mirror to layer suffix \_BOTTOM and vice versa). When padstack user mask layers was introduced in 16.2, they did not offer this capability. New designs will offer this capability but designs containing user mask layers that were created in 16.2 or 16.3 will have this mirror option disabled. a cdsz file.

Symphony Supported: No

Name: *xnetState*  
Value: dml, gates, or off  
Set?: No  
Description: Queries current database xnet state  
Equiv: none  
Side Effects: none  
Symphony Supported: Yes



*Tip*  
The first argument is a symbol ('drcEnable) and not a string ("drcEnable"). See [axlBackdrillGet](#) to get backdrill status.

## Arguments

<i>s_name</i>	Symbol name of control. nil returns all possible names.
<i>g_value</i>	Optional symbol value to set. Usually a t or a nil.

## Value Returned

See above.

<i>ls_names</i>	If name is nil, then returns a list of all controls.
-----------------	--

## Example

- Sets DRC system off

```
old = axlDBControl('drcEnable, nil)
```

- Gets current value of pin-2-pin ratsnest

```
current = axlDBControl('ratsnestDistance)
```

- Gets all names supported by this interface

```
listOfNames = axlDBControl(nil)
```

- Set dynamic shape mode to smooth

```
old = axl0DBControl('dynamicFillMode, 'wysiwyg)
```

## See Also

[axlDBDisplayControl](#)

## 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`.

If a list of `dbids` is provided, the union of the extents is returned.

Unsupported dbids are ignored. Dbids that contribute to the extent are SEGMENT, CLINE, LINE, RECTANGLE, FRECTANGLE, FIGURE, TEXT, SHAPE, VOID, VIA, PIN, and SYMBOL\_INSTANCE.

### Arguments

<code>o_dbid</code>	dbid of an element.
<code>lo_dbid</code>	list of dbids
<code>visible_only</code>	t return the extents of visible parts of the element. nil return the extents of the entire element.

### Value Returned

<code>bBox</code>	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).
<code>nil</code>	Error; bad arguments

## **axlDBIgnoreFixed**

```
axlDBIgnoreFixed(  
    [g_ignore]  
) -> t/nil
```

### **Description**

Provides similar functionality to that offered by many Allegro batch programs which allow FIXED and LOCKED properties to be ignored (for example: netrev -z).

If g\_ignore is t, FIXED testing is ignored; nil restores FIXED testing. No argument reports the current state of FIXED testing.

This does not disable READ-ONLY states. A parent can be locked, which means the children cannot be modified (symbols locked to prevent editing of its components). READ-ONLY objects are typically seen when you are in the partition editor.



***It is important that FIXED property testing be restored. Allegro automatically restores FIXED testing if your SKILL program returns to Allegro. This includes calling axlShell.***

**Note:** Recommend using axlDBCloak since that API catches any SKILL errors and restores the mode.

### **Arguments**

g\_ignore            If t, turn off FIXED testing; nil restore.

no argument        Return current state of FIXED testing.

### **Value Returned**

old fixed (nil FIXED is enable, t is enabled)

### **See Also**

[axlDBIsFixed](#), [axlDBCloak](#)

## Examples

```
; Select an object  
p = ashOne()  
  
; Add fixed property  
axlDBAddProp(p, '("FIXED" t))  
  
; Attempt to delete it  
axlDeleteObject(p)  
  
; now delete it  
axlDBIgnoreFixed(t)  
axlDeleteObject(p)  
axlDBIgnoreFixed(nil)
```

## **axlDBIsReadOnly**

```
axlDBIsReadOnly(  
    o_dbid  
) -> t/nil
```

### **Description**

This API command checks if indicated database object is read-only.

An example of why an object is marked read-only due to a partition being active.

### **Arguments**

*o\_dbid* dbid of the element to be checked.

### **Value Returned**

- *t* – object is read-only
- *nil* – not read-only or not a dbid

### **See Also**

[axlDBIsFixed](#)

### **Example**

```
p = axlSelectByname("SYMBOL" "U1")  
ret = axlDBIsReadOnly(p)
```

## **axIDBSectorSize - Obsolete**

This is obsolete. It is kept for backwards compatibility. It now calls [axIDBTuneSectorSize](#). Tuning sectors is now built into dbdoctor.

## **axlGetDrawingName**

```
axlGetDrawingName(  
    )  
⇒ t_drawingPathName
```

### **Description**

Retrieves the full path of the drawing.

### **Arguments**

None

### **Value Returned**

*t\_drawingPathName* Full path of the drawing.

### **See Also**

[axlCurrentDesign](#)

### **Example**

```
axlGetDrawingName() => "/net/nile/home/neeti/testboards/test1.brd"
```

## **axlIgnoreFixed**

See [axlDBIgnoreFixed](#).

## **axlInTrigger**

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

### **Description**

Tests if the application or the utility in a axlTrigger callback. Returns `t` if currently in a trigger call. This function is only required in rare cases when you might have a utility that you want to run differently if called as part of a axlTrigger callback.

### **Arguments**

None

### **Value Returned**

`t` if in a trigger callback, `nil` otherwise

### **See Also**

[axlTriggerSet](#), [axlInTriggerFunc](#)

## **axlInTriggerFunc**

`axlInTriggerFunc() -> s_triggerFunc/nil`

### **Description**

If in a trigger callback, reports trigger function symbol. `nil`, if not in a trigger callback.

### **Arguments**

No argument required

### **Value Returned**

*s\_triggerFunc*      Current function name if in a trigger or `nil` if not in a trigger

### **See Also**

[axlTriggerSet](#), [axlInTrigger](#)

## **axlIsSymbolEditor**

`axlIsSymbolEditor() -> t/nil`

### **Description**

Returns `t` if in symbol editor, `nil` for all other editors.

When loading custom menus you might want to differentiate between the symbol editor and design editor.

### **Arguments**

None.

### **Value Returned**

Returns `t` if symbol editor.

### **See Also**

[axlDesignType](#)

### **Example**

`axlIsSymbolEditor()`

## **axlKillDesign**

axlKillDesign()  
⇒ t/nil

### **Description**

Same as (axlOpenDesign <unnamed> "wf"), where <unnamed> is the standard Allegro 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.

### **Note**

This will void all *axl dbid* handles and clear the selection set.

### **See Also**

[axlOpenDesign](#)

## axlOpenDesign

```
axlOpenDesign(  
    ?design t_design  
    ?mode t_mode  
    ?noMru g_noMru  
    ?ignoreLock g_noMru  
)  
⇒ t_design/nil
```

### Description

Opens a design. The new design replaces the current design. If the current design has unsaved edits, confirmation is needed for discarding the edits, unless the current design was opened in "r" mode (not supported in this release) or *t\_mode* is "wf" or "nf". If cancelled, the function returns *nil* and the current design is left remains open.

If *t\_mode* is "n", opens a new design and specify the name and location on disk for that design without having it read any existing design of that name at that location.



**Warning:** Any existing design of that name at that location will be overwritten when the design is saved.

If *t\_design* is not provided, you are prompted for the name of the design to open.

If *t\_design* is provided but does not exist on disk, the standard Allegro PCB Editor Drawing Parameters form opens.

The designType of the new design is set as the same as the current design. It can be changed using axlSetDesignType.

## Arguments

<i>t_design</i>	The name of the new design to edit
<i>t_mode</i>	Can be any one of "w", "wf", or "n". Add "l" to disable Allegro file locking.  The "w" mode opens a design in the write mode, "wf" in the forced write mode, and "wl" in no lock mode.
<i>g_noMru</i>	If set to t, the design is not included in the most recently used file list. (Default is to update.)
<i>ignoreLock</i>	If set to t, design lock file is ignored if present.  This is not recommended by Cadence.

## Value Returned

<i>t_design</i>	If design opens
<i>nil</i>	If error.

**Note:** This function is similar to the Allegro PCB Editor `open` command, except that the *t\_mode* can be set to "wf" in order to discard the current edits without confirmation. However, when *t\_mode* is 'n', it behaves the same as the Allegro PCB Editor `new` command.

Since confirmation uses the standard Allegro PCB Editor confirmers, using the "wf" mode is essentially the same as setting the `NOCONFIRM` environment variable.

This will void all `ax1_dbid` handles and clear the selection set.

## See Also

[ax1CurrentDesign](#), [ax1KillDesign](#), [ax1RenameDesign](#), [ax1SaveDesign](#),  
[ax1SetSymbolType](#), [ax1RunBatchDBProgram](#), [ax1SaveEnable](#),  
[ax1CompileSymbol](#)

## **axlOpenDesignForBatch**

```
axlOpenDesignForBatch(  
    t_design  
    t_mode  
) ==> t_design/nil
```

### **Description**

Opens a design. The new design replaces the current design. Since this is for batch usage, commands and menus are not changed. If the current design has unsaved edits, then you are asked to confirm the discard unless the current design was opened in "r" mode (not supported in this release) or g\_mode is "wf". If you cancel the confirmmer, then the function returns nil and the current design is left open.

If t\_design is not provided, then you are prompted for the name of the design to open.

If t\_design does not exist on the disk, then the standard Allegro Drawing Parameters form appears.

The designType of the new design is set the same as the current design. It can be changed using axlSetDesignType.

### **Arguments**

t_design	The name of the new design to edit
g_mode	"w" or "wf" (see above)
	add a "l" to disable Allegro file locking
	example: "wl" for no locking

### **Value Returned**

t\_design if opened, or nil if error

**Note:** This command functions the same as the Allegro edit command, except that the t\_mode can be set to "wf" in order to discard the current edits without confirmation. Since confirmation uses the standard Allegro confirmmer, the "wf" mode is the same as setting the NOCONFIRM environment variable.

This removes all axl dbid handles and clears the selection set.

## Example

Open u.dra in current directory with noconfirm option

```
axlOpenDesignForBatch("u.dra" "wf")
```

## **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 (without file extension).

### **Value Returned**

*t\_design*                    New current design name.

*nil*                        Error due to incorrect argument.

### **See Also**

[axlOpenDesign](#)

### **Examples**

```
axlRenameDesign("foobar")
```

## **axlSaveDesign**

```
axlSaveDesign(  
    ?design t_design  
    ?mode t_option  
    ?noMru g_noMru  
    ?noConfirm g_noConfirm  
    ?writeModel g_write  
)  
⇒ 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*, you are 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.

### **NOTES**

- This is essentially the Allegro "save" command.
- This will clear all axl dbid handles and the selection set.
- Use axlRunBatchDBProgram, if the intent is to save the design to run another program.

## Arguments

<i>t_design</i>	Name for the saved design.
<i>t_option</i>	"nocheck" - No database check performed.
<i>g_noMru</i>	If <i>t</i> does not update the Most Recently Used file list. (Default is to update.)
<i>g_noConfirm</i>	If <i>t</i> does not present a confirm if overwriting an existing design. File is overwritten.
<i>g_write</i>	If <i>t</i> uses allegro write file model. This means file is written to disk with provided name but current drawing name is not updated.

## Value Returned

<i>t_design</i>	Name for the saved design.
nil	Error due to incorrect argument(s).

**Note:** Essentially the Allegro PCB Editor *save* command. The current design name is not changed. Use *axlRenameDesign* to change the current design name.

This will void all *axl dbid* handles and clear the selection set.

## See Also

[axlOpenDesign](#), [axlRenameDesign](#), [axlRunBatchDBProgram](#)

## **axlSaveEnable**

```
axlSaveEnable(  
    [g_saveEnable]  
) -> t/nil
```

### **Description**

This queries or sets the design to save design. When `t`, the *File – Save design* menu command is enabled. If `nil`, it is disabled.

If no arguments are provided, then returns the current status. If `g_saveEnable` is `t`, then enables save menu (reset of next database save). If `g_saveEnable` is `nil`, disables the save menu.

It is not recommended that you unset the save enable.

### **Arguments**

`g_saveEnable`      `t`

If not provided, just does query.

### **Value Returned**

Returns current save setting if query and previous setting if changing the value.

### **See Also**

[axlSaveDesign](#)

### **Examples**

#### **1) Query state of save**

```
state = axlSaveEnable()
```

#### **2) Set state of save to t (enable save menu)**

```
oldState = axlSaveEnable(t)
```

## **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_dbuperuu/nil]  
    x_drcCount/nil  
)  
⇒ x_drcCount/nil
```

### **Description**

Changes the units, accuracy, or dbuperuu of the design. Either accuracy or dbuperuu can be set. Both cannot be set at the same time. 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.

<b>Units</b>	<b>Min Accuracy</b>	<b>Max Accuracy</b>	<b>Delta</b>
mils	0	4	0 (more than 2 not advised due Gerber)
inches	2	4	3
microns	0	4	0
millimeters	1	4	3
centimeters	2	4	4

- Decreasing accuracy is not recommended.
- Switching units between Metric and English is not recommended due to inevitable rounding issues.
- 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
```

## Allegro SKILL Reference

### Design Control Functions

---

- 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 MILS 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 MILS 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.

### **DBUperrUU Notes**

- Only use DBUperrUU to match another design tool, which supports DBUperrUU (such as Virtuoso, designs coming from OA, and so on).
- When changing units, you should change either accuracy or DBUperrUU to maintain adequate precision within the database.
- The value of DBUperrUU chosen will dictate the internal accuracy.

### **Examples**

DBUperrUU	Accuracy
200	3
2000	4
4000	5

- Legal values for dbuperuu are: 160, 200, 400, 800, 1600, 2000, 4000, 8000

## Arguments

<i>t_units</i>	Mils, inches, millimeters, centimeters, or microns.
<i>x_accuracy</i>	Range is 0 to 4, according to the table shown.
<i>x_dbuperuu</i>	160, 200, 400, 800, 1600, 2000, 4000, 8000

## Value Returned

<i>x_drcCount</i>	DRC count
<i>nil</i>	Failed to change design units
<i>nil nil nil</i>	String of legal dbuperuu values

## Notes:

- This function may take extra time on large designs.
- This function reruns DRC if enabled.

## Example 1

```
axlDBChangeDesignUnits("millimeters" 4)
```

Changes a design from mils/0 to millimeter/4 (accuracy maintains database precision).

## Example 2

```
axlDBChangeDesignUnits(nil 2)
```

Increases accuracy from 0 to 2 and keeps the same units.

## Example 3

```
axlDBChangeDesignUnits(nil nil 4000)
```

Set dbuperuu 4000 to match Virtuoso and keep 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
<code>'general</code>	Performs a full database check.
<code>'links</code>	Performs a full database check.
<code>'branch</code>	Performs a full database check.
<code>'shapes</code>	Checks shapes (autovoid) <ul style="list-style-type: none"><li>■ May be slow for complicated shapes.</li><li>■ Normally fast.</li><li>■ Slow on large databases.</li></ul>
<code>'all</code>	Performs a full check and fix. Does not perform a shape check.
<code>'drc</code>	Performs a batch DRC.
<code>'log</code>	Uses the standard <code>dbdoctor.log</code> 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(
    o_dbid/t_padstackName
    lt_startEnd
    [g_dontTrim]
)
⇒ o_dbid/nil
```

### Description

Creates a new padstack by copying an existing padstack, with optional removal of layers. The name for the new padstack automatically derives from the existing name by adding a unique post fix. The padstack is marked in the design as derived from its starting padstack.

Start/end must be provided and must be legal ETCH layers.



**If the padstack is not used, it is deleted by the next dbdoctor run.**

### Arguments

<i>o_dbid</i>	<i>dbid</i> of the padstack to copy from.
<i>t_padstackName</i>	Name of the padstack to copy from.
<i>lt_startEnd</i>	List of start ( <i>class/subclass</i> ) and stop ( <i>class/subclass</i> ) layers. If class is used, it must be ETCH (PCB) or CONDUCTOR (APD+)
<i>g_dontTrim</i>	Obsolete, ignored.

### Value Returned

<i>dbid</i>	<i>dbid</i> of the new padstack.
<i>nil</i>	Failed to create new padstack.

## **Allegro SKILL Reference**

### Design Control Functions

---

#### **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, '("TOP" "2") nil)
```

or

```
newPadId = axlDBCopyPadstack("VIA", '("TOP" "GND") nil)
```

## **axIDBDelLock**

```
axIDBDelLock(  
    [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.

View locks cannot be unlocked.

### **Arguments**

*t\_password*      Optional password string.  
                        String length should be 6 to 20 characters.  
                        Period (.), space, forward slash (\) and some character patterns  
                        are not legal.

### **Value Returned**

<i>t</i>	Lock is removed or there is no lock to remove.
<i>nil</i>	Failed to remove lock due to an incorrect password.

### **See Also**

[axIDBSetLock](#), [axIDBGetLock](#)

### **Example**

- Delete lock with no password.  

```
axIDBDelLock()
```
- Delete lock with the password, "foobar".  

```
axIDBDelLock("foobar")
```

## Allegro SKILL Reference

### Design Control Functions

---

## axIDBGetLock

```
axIDBGetLock()  
⇒ 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 (auto-generated)
<i>t_lockDate</i>	Date the database was locked (auto-generated)
<i>t_systemName</i>	System on which the database was locked (auto-generated)
<i>t_locklevel</i>	Level of the lock: view, export or write
<i>t_expiration</i>	Expiration duration associated with the lock from the lock date
<i>t_comments</i>	Comments set by user who locked database
<i>g_useNtp</i>	<i>t</i> if using a NTP for timer, <i>nil</i> uses system time
<i>t_ntpServer</i>	NTP server name or null

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

<i>nil</i>	Database is unlocked.
<i>l_info</i>	List ( <i>t_userName t_lockDate t_systemName t_locklevel t_expiration t_comments g_useNtp t_ntpServer</i> )
	Database is locked.

### See Also

[axIDBSetLock](#), [axIDBDelLock](#)

## **axIDBMemoryReclaim**

```
axIDBMemoryReclaim( )  
    -> x_sizeReclaimed
```

### **Description**

Reclaims database memory for reuse by the Allegro database. Normally in SKILL memory of deleted database objects is not reused until SKILL code returns to the main processing loop.

Should only be used in special cases since the default programming model works for most all SKILL programs. For a well written SKILL program this is typically not required.

Before utilizing this API, first try the following techniques:

- use axIDBCloak if are adding/deleting objects that are etch based (vias, clines, etch shapes, pings (for example, like moving a symbol)).
- insure that you do not have any long running db transactions (axIDBTransactionStart). For example, commit all transactions you have active. Transactions can be nested so you should commit all the way to the initial transaction.

The one known programming model that uses this API is the "try-it" model. This is where a program adds an object to the database, performs a test and then deletes it (for example, utilizes axlAirGap to get distances to other db objects).

For the API to be most effective make sure:

- Are not inside an axIDBCloak.
- The most memory can be reclaimed if you have no active db transactions.

Memory reclaimed is returned to the database for reuse. It is not be returned to the program's memory pool nor returned to the OS. The program's memory usage at the OS level is NOT reduced.

As a side effect some of the dbids that are checked-out may be reclaimed since they actually link to deleted db objects. They are reported as "dbid:remove".

**Note:** Do NOT use instead axl trigger callbacks.

### **Arguments**

none

**Value Returned**

Amount of memory reclaimed (bytes).

**See Also**

[axIDBCloak](#), [axIDBTransactionStart](#)

## axIDBSetLock

```
axIDBSetLock(
  ['password t_password]
  ['locklevel           t_locklevel]
  ['expiration         t_expiration/x_expiration]
  ['comments t_comments]
  ['useNtp              t/nil]
  ['ntpServer          t_server]
)
⇒ t/nil
```

### Description

Provides a mechanism to lock the database against future changes. If you lock the database it is locked in memory, to save the lock, you need to save the database to disk. Therefore, after locking the database, you can save the database once before all further attempts to save the database are rejected.

When a user opens a locked database, a warning, indicating that the database is locked, is thrown. An attempt to save the database fails with the above exception of above and uprev (see following text).

### Arguments

password	If provided, this must be provided to unlock the database. If you forget the password you will not be able to unlock the database. <ul style="list-style-type: none"><li>■ Password must be greater than 5 or less than 20.</li><li>■ Following characters are not allowed in the password.<ul style="list-style-type: none"><li>○ \ whitespace</li><li>○ leading dash (-)</li></ul></li></ul>
locklevel	Three levels of locking are provided with the lock — view, export, and write. If user does not provide any locklevel, or provides the wrong argument, or provides it without any password, default level of write would be used.
expiration	With the lock, user can set the expiration duration for the design. The arguments passed must be in number of days with a minimum of 1. If wrong expiration is provided, then default (no duration limit) is used.

## Allegro SKILL Reference

### Design Control Functions

---

comments	Free form comments. You may embed carriage returns ("\\r") in the string to force a new-line in the locking user interface.
useNtp	Use NTP server to verify time when a expiration value is provided
ntpServer	NTP Server to use for verify an expiration lock. If not provided uses NTPSERVER environment variable which defaults to 0.pool.ntp.org.



***If using NPT based expiration locks then you must ensure that the user opening the design can access the NTP server specified. Frequently, firewalls block the NTP serve/port. The port is 123 UDP.***

**Note:** Upgrading the database from to a new version ignores the lock flag. For a simple lock provide no arguments. If a nil is provided, it returns list of options to this function.

#### Value Returned

t	Locked database.
nil	Failed to lock database due to argument errors or database already locked

#### Examples

- Most basic lock (no password, no locklevel, no expiration, no comments)

```
axlDBSetLock()
```

- Lock with comments and disable data export

```
axlDBSetLock('password "passwd" 'locklevel "view" 'expiration 6 'comments  
"Test")  
axlSaveDesign(?design "locked_data")  
printf("Locked info %L\n" axlDBGGetLock())
```

## **Allegro SKILL Reference**

### Design Control Functions

---

#### **See Also**

[axIDBGetLock](#), [axIDBDelLock](#), [axISaveDesign](#)

## **axlDBTuneSectorSize**

```
axlDBTuneSectorSize( )
  ==> nil/l_result
```

### **Description**

This tune's Allegro's sector size for better performance. Normally, this occurs automatically via dbdoctor or performance advisor. In addition, it is done periodically during design open.

Allegro's optimal sector size changes over the course of a design cycle. As the design becomes complete a smaller sector size typically results in better performance at a cost of a slightly higher memory requirement.

**Note:** For the current release, the sector size is in dbunits, future releases may change this to design units.

### **Arguments**

None

### **Value Returned**

- **nil:** If no tuning is required
- a disembody property list: If tuning was required. The a disembody property list contains:
  - old sectors
  - their size
  - new sectors
  - new sector size

## **axlTechnologyType**

axlTechnologyType()  
⇒ *t\_technology*

### **Description**

Returns the type of design technology in use.

### **Arguments**

none

### **Value Returned**

*t\_technology*      Technology type as shown:

#### **Technology Type**

#### **Product**

"mcm"

Allegro Package Designer L or Allegro Package SI XL

"pcb"

Allegro 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**

<i>s_trigger</i>	Trigger type. See <code>axlTriggerSet</code> .
<i>s_function</i>	Trigger function to remove - the same as you passed to <code>axlTriggerSet</code> .

### **Value Returned**

<code>t</code>	Trigger removed.
<code>nil</code>	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 Allegro PCB Editor. Supported events are shown by the *s\_trigger* option. 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. So it is recommended that you must keep these 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.

## Allegro SKILL Reference

### Design Control Functions

---

- Allegro PCB Editor sends a close trigger when Allegro PCB Editor exits normally. Abnormal exits such as crash, user kill, etc. result in this 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.
- You should use caution when using `axlShell` API within a trigger function. Cadence advises against this and does not support it. For example, calling Allegro commands or running scripts is not supported. Using `axlShell` API within a trigger function can cause the following:
  - Allegro layout editor may block in the script resulting in trigger failures.
  - Messages may appear to the user that you cannot suppress.
  - Scripts, commands and command behavioral can change from release to release.
  - User can also issue commands to different functionality.
- The triggers for opening and saving the database are generally not called when the database needs to do 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

*s\_trigger* Trigger type as follows:

<b>s_trigger Option</b>	<b>Description</b>
'open ( <i>t_database</i> <i>g_existing</i> )	Called immediately after a database is opened. Passed a list of two items: the name of the database and <i>t</i> if existing or <i>nil</i> if a new database.  Restrictions: You should not open, save, and close the current database.
'save ( <i>t_oldName</i> <i>t_newName</i> )	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.
'close <i>t_name</i>	Called before another database is opened. The single item is the name of the database being discarded.
'exit <i>x_status</i>	Called when program is exiting except if it is an abnormal termination.  <i>x_status</i> is a number where 0 indicates a clean exit, 1 indicates warnings, and 2 is exiting due to an error.

**Note:** GUI programs do not currently exit with warnings but this may change in the future.

Trigger is provided for applications to clean-up the environment. For database specific work, use save or close trigger.

Restrictions:

- Do not read or change the database.
- Do not display dialogs or blocking confirmers.

## Allegro SKILL Reference

### Design Control Functions

---

<b>s_trigger Option</b>	<b>Description</b>
'menu t_menuName	<p>Called when a new menu is loaded in the main tool window.</p> <p>Targeted at application code to modify the menu.</p> <p>Restrictions:</p> <ul style="list-style-type: none"><li>■ While a database is active do not change the database.</li><li>■ Do not display dialogs or blocking confirmers.</li><li>■ Do not call axlUIMenuLoad.</li></ul>
'xprobe (s_mode lo_dbid)	<p>Called when object(s) is highlighted or dehighlight <i>s_mode</i> is a symbol which may be highlight or dehighlight and <i>lo_dbid</i> is a list of dbids. This is the same interface used to cross-probe to ConceptHDL/Capture.</p> <p>This is targeted to allow sending messages to external tools.</p> <p>Restrictions:</p> <ul style="list-style-type: none"><li>■ Do not change the database</li><li>■ Do not invoke a dialog or blocking confirmers</li><li>■ Keep processing at a minimum since this will impact user interactive performance.</li></ul>
'window l_windowBox	<p>Called when the canvas display window changes the display area. This may be caused by a zoom, a pan in any direction, or a resize of the canvas window. <i>l_windowBox</i> is a bBox giving the current display area of the database.</p> <p>This is not triggered during a roam (mouse panning) action. This would flood your trigger with events as the screen is being constantly repositioned as your user moves the mouse.</p> <p>Restrictions:</p> <ul style="list-style-type: none"><li>■ Do not change the window display region in this trigger.</li><li>■ This can cause infinite loops .</li></ul>
<i>s_function</i>	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.

## See Also

[axlTriggerClear](#), [axlTriggerPrint](#), [axlInTrigger](#), [axlInTriggerFunc](#)

## 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

```
when( isCallable('axlTriggerSet) axlTriggerSet('open 'MyTriggerOpen))
```

To be compatible with pre-14.1 software, substitute for axlTriggerSet in allegro.ilinit.

## **axlGetActiveLayer – Obsolete**

This function is obsolete and is kept for compatibility reasons. Use `axlDBControl`.

## **axIGetActiveTextBlock – Obsolete**

This function is obsolete.

## **axlSetActiveLayer – Obsolete**

This function is obsolete and is kept for compatibility reasons. Use `axlDBControl`.

## **axlWFMAnyExported**

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

### **Description**

Reports if there are any exported partitions.

Use axlDesignType to see if the design is currently a partition.

### **Arguments**

None

### **Value Returned**

t, if one or more partitions are currently exported

nil, if no partitions exported

### **See Also**

[axlDesignType](#)

### **Examples**

```
axlWFMAnyExported()
```

## **axIDBDisplayControl**

```
axIDBDisplayControl(  
    s_name  
    [g_value]  
)  
==> g_currentValue/ls_names
```

### **Description**

This command is used to inquire and set the display database controls. When used for setting a value, the command returns the old value of the control.

**Note:** For most of these controls, if the form that is displaying the current setting is active, it may not be updated. Additional side effects of individual controls are listed.

Use the [axIColorGet](#) and [axIColorSet](#) commands to change the background color.

Items currently supported are listed in [Table 14-1](#) on page 1022.

**Table 14-1 Supported Controls**

Name	Value	Set	Description	Equivalent	Side Effects
connectPointSize	dbrep	Yes	Changes the size of connect points (diamond figures)	prmed Display group	Call <a href="#">axIVisibleUpdate</a> to update display.
connectPointEnable	t/nil d	Yes	Changes visibility of connect points (diamond figures)	prmed Display group	Call <a href="#">axIVisibleUpdate</a> to update display.
customColorEnable	t/nil d	Yes	Changes the display of custom colors	color192 dialog ("Enable Custom Colors")	Call <a href="#">axIVisibleUpdate</a> to update display.
displayNetNames	t/nil	Yes	Enables the display of net names on etch. OpenGL must be enabled.	prmed Display group	Call <a href="#">axIVisibleUpdate</a> to update display.

## Allegro SKILL Reference

### Design Control Functions

---

**Table 14-1 Supported Controls, *continued***

Name	Value	Set	Description	Equivalent	Side Effects
designOrigin	t/nil	Yes	Enables the display of design origin	prmed Display group	Call <a href="#">axlVisibleUpdate</a> to update display.
drcMarkerSize	dbrep	Yes	Changes the size of DRC markers	prmed Display group	Call <a href="#">axlVisibleUpdate</a> to update display.
endcapsEnable	t or nil	Yes	Controls display of line endcaps	prmed Display group	Call <a href="#">axlVisibleUpdate</a> to update display.
filledPadsEnable	t or nil	Yes	Pads are displayed filled or hollow	prmed Display group	Call <a href="#">axlVisibleUpdate</a> to update display.
gridColor	1 to < <i>maxColor</i> >	Yes	Changes the grid color	color dialog Display group	Call <a href="#">axlVisibleUpdate</a> to update display.
			<b>Note:</b> You can find the < <i>maxColor</i> > by running the command, <i>maxColor</i> = axlColorGet(`count)		
gridEnable	t or nil	Yes	Queries and changes the grid visibility	Color form, Display Group, Grids	Call <a href="#">axlVisibleUpdate</a> to update display.

## Allegro SKILL Reference

### Design Control Functions

---

**Table 14-1 Supported Controls, *continued***

Name	Value	Set	Description	Equivalent	Side Effects
highlightColor	1 to <code>&lt;maxcolor&gt;</code>	Yes	Queries/changes the permanent highlight color. Can be used with <code>axlHighlightObject</code>	Color form, Display Group, Permanent highlight	Also changes the perm highlight color in the <code>hilite</code> command.  Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
holeColor	1 to <code>&lt;maxcolor&gt;</code>	Yes	Queries/changes the drill hole color.	Color form, Display Group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
backdrillHoleColor	1 to <code>&lt;maxcolor&gt;</code>	Yes	Queries/changes the backdrill hole color.	Color form, Display Group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
lastSaveUser	string	No	Reports last user login who saved design. This may be an empty string, if the design was never saved.	Status form	None
nonPlatedHolesEnabled	t or <code>nil</code>	Yes	Queries and changes the non-plated holes visibility.  Unlike in the <code>prmed</code> dialog, setting this option to t does not set <code>padlessHolesEnable</code> to t.	Color form, Display Group, Grids	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.

## Allegro SKILL Reference

### Design Control Functions

---

**Table 14-1 Supported Controls, *continued***

Name	Value	Set	Description	Equivalent	Side Effects
backdrillHolesEnable	t or nil	Yes	Queries and changes the backdrill holes visibility.  Must have performed the backdrill process to see the holes.	Color form, Display Group, Grids	Call <a href="#">axlVisibleUpdate</a> to update display.
padlessHolesEnable	t or nil	Yes	Queries and changes the padless holes visibility	Color form, Display Group, Grids	Call <a href="#">axlVisibleUpdate</a> to update display.
platedHolesEnable	t or nil	Yes	Queries and changes the plated visibility.  Unlike in the prmed dialog, setting this option to t does not set padlessHolesEnable to t.	Color form, Display Group, Grids	Call <a href="#">axlVisibleUpdate</a> to update display.
tempColor	1 to <maxcolor>	Yes	Queries/changes the temporary highlight color.  Can be used with <code>axlHighlightObject</code>	Color form, Display Group, Temporary highlight	Call <a href="#">axlVisibleUpdate</a> to update display.
ratsnestColor	1 to <maxcolor>	Yes	Queries/changes the ratsnest color for top to bottom ratsnest. In pre-16.2 releases, this set the color for all ratsnest.	Color form, Display Group, Ratsnest Color	Call <a href="#">axlVisibleUpdate</a> to update display.

## Allegro SKILL Reference

### Design Control Functions

---

**Table 14-1 Supported Controls, *continued***

Name	Value	Set	Description	Equivalent	Side Effects
ratsnestBBColor	1 to <code>&lt;maxcolor&gt;</code>	Yes	Queries/changes the ratsnest color for bottom to bottom ratsnest.	Color form, Display Group, Ratsnest Color	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
ratsnestTTColor	1 to <code>&lt;maxcolor&gt;</code>	Yes	Queries/changes the ratsnest color for top to top ratsnest.	Color form, Display Group, Ratsnest Color	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
ratsnestJog	t/nil	Yes	Queries/changes the ratsnest jog option.	prmed form, Display Group, Ratsnest Jog	None
ratTSize	dbrep	Yes	Changes the size of RatT markers	prmed Display group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
thermalPadEnable	t/nil	Yes	Queries and changes the thermal pads. Only applicable for negative planes.	prmed Display group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
transparencyGlobal	1 to 255	Yes	Changes the OpenGL global transparency where 1 is completely translucent and 255 is solid.	prmed Display group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.

## Allegro SKILL Reference

### Design Control Functions

---

**Table 14-1 Supported Controls, *continued***

Name	Value	Set	Description	Equivalent	Side Effects
transparencyShape	1 to 255	Yes	Changes the OpenGL shape transparency where 1 is completely translucent and 255 is solid.	prmed Display group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
viaLabel	t/nil	Yes	Queries and changes the via label display. Now called "Drill labels" in dialog	prmed Display group Now called "Drill labels" in dialog	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
viaLabelColor	1 to <maxcolor>	Yes	Queries and changes the via label color. Now called "Drill labels" in dialog	Color form, Display Group Now called "Drill labels" in dialog	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
stackedViaLabelColor	1 to <maxcolor>	Yes	Queries and changes the stacked via label color.	Color form, Display Group	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
waiveDRCColor	1 to <maxcolor>	Yes	Queries and changes the waived DRC color.	Color form, Display Group, Waived DRC	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.
waiveDRCEnable	t/nil	Yes	Queries and changes the waived DRC display state	Color form, Display Group, Waived DRC	Call <a href="#"><u>axlVisibleUpdate</u></a> to update display.

## Arguments

<i>s_name</i>	symbol name of control. nil returns all possible names
<i>s_value</i>	optional symbol value to set. Usually a t or a nil.

## Value Returned

See above

`ls_names`: If name is nil then returns a list of all controls.

## See Also

[axlDBControl](#), [axlColorGet](#), [axlColorShadowGet](#)

## Examples

- Find out grid color  
`color = axlDBDisplayControl('gridColor, nil)`
- Turn on grids  
`old = axlDBDisplayControl('gridEnable t)`
- Get all names supported by this interface  
`listOfNames = axlDBDisplayControl(nil)`

---

# Database Create Functions

---

## Overview

This chapter describes the AXL functions that add objects to the Allegro PCB Editor database. Some functions require input that you set up using available auxiliary functions, which are also described in this chapter. For example, Allegro PCB Editor paths consist of any number of contiguous line and arc segments. To add this multi-structure to the Allegro 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 Allegro 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 (`DBCreate`) functions modify the active Allegro 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 Allegro 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 `DBCreate` functions create are called *figures*. `DBCreate` functions return, in a list, the *dbids* of any figures they create and a Boolean

## Allegro SKILL Reference

### Database Create Functions

---

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 (Allegro PCB Editor class/subclass) by calling the `ax1SetCurrentLayer` 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 Allegro 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. Allegro 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.

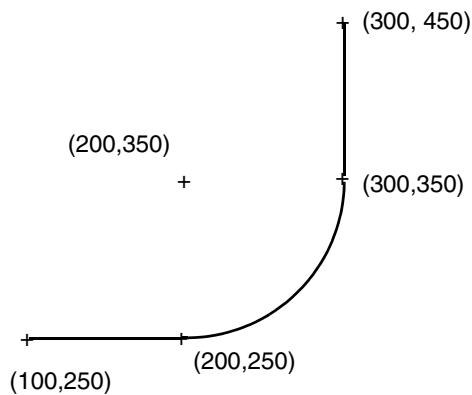
A `path` is a set of contiguous arc and single straight line segments. In AXL, you first create a path consisting of the line and arc segments by adding each segment with a separate AXL function, then creating the actual figure using the appropriate `ax1DBCreate` function, with the path as one of the 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, 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

#### Value Returned

<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.
<i>r_path/nil</i>	Returns the <i>r_path</i> handle.

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

### Example

See start of the [Path Functions](#) on page 1030.

## **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 15-1](#) on page 1034.

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

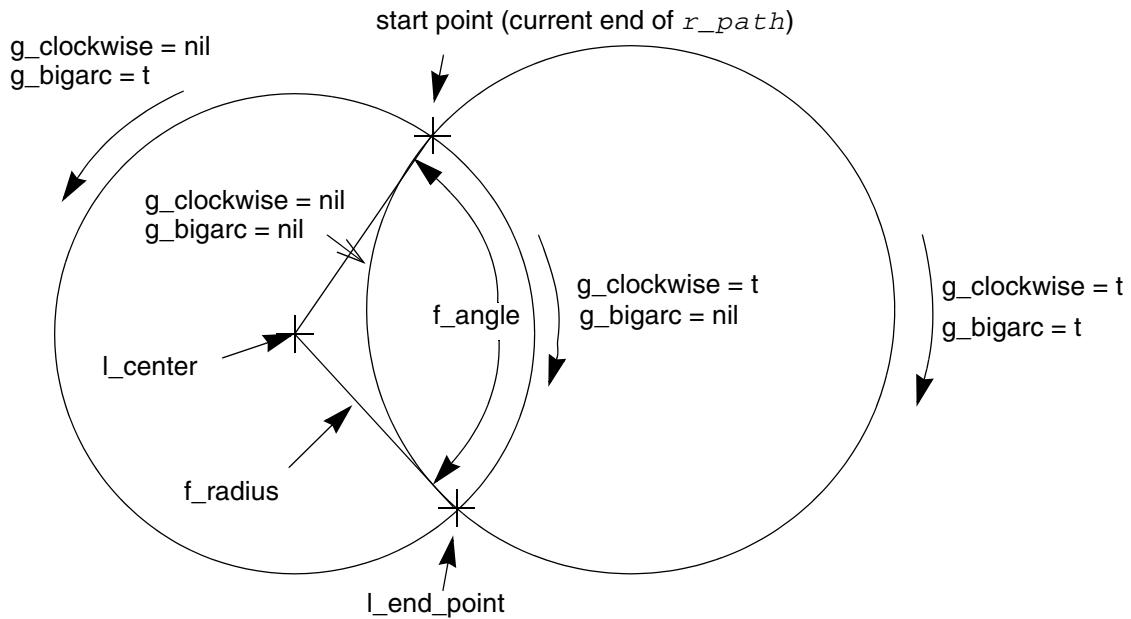
## Arguments

### Figure 15-1 Effects of *axlPathArc* 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 <i>axlPathStart;nil = use current width</i>
<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: <i>t</i> → create arc clockwise from start to endpoint <i>nil</i> → create counterclockwise. Default is counterclockwise (See <a href="#">Figure 15-1 on page 1034</a> ).
<i>g_bigarc</i>	<i>axlPathArcRadius</i> : Create an arc greater than or equal 180 degrees (See <a href="#">Figure 15-1 on page 1034</a> ).
<i>f_radius</i>	<i>axlPathArcRadius</i> : Arc radius in user units.
<i>f_angle</i>	<i>axlPathArcAngle</i> : Angle in degrees subtended by arc (See <a href="#">Figure 15-1 on page 1034</a> ).
<i>l_center</i>	<i>axlPathArcCenter</i> : Arc center point in absolute coordinates.

## Allegro SKILL Reference

### Database Create Functions



#### Value Returned

<i>r_path</i>	Current path handle.
<i>nil</i>	Arc path not created.

#### Example 1

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

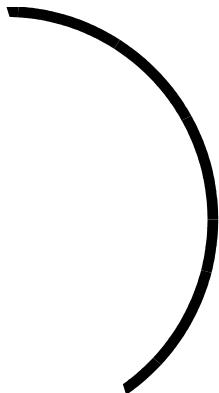
Adds a smaller-than-180 degree counterclockwise arc by radius.

## Allegro SKILL Reference

### Database Create Functions

---

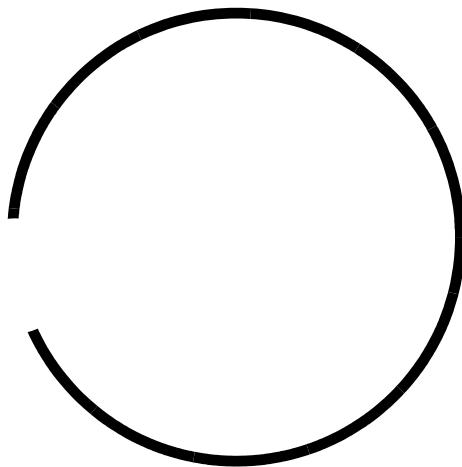
Creates the smaller possible arc:



### 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 1031.

## 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_rath</i> structure.
<i>nil</i>	No line segment added to <i>r_path</i> structure.

### Example

See start of the [Path Functions](#) on page 1030.

## 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

<i>f_width</i>	Default width of the path structure.
nil	<i>r_path</i> 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**

### **Example**

*r\_pathList* Returns a list of the segments of the path.

*nil* *r\_path* is not a path.

```
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**

### **Example**

*r\_pathList* Returns the last segment of the path.  
*nil* *r\_path* is not a path.

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**

<i>l_point</i>	List containing the center coordinate of the arc segment.
<i>nil</i>	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**

<i>t</i>	Segment is clockwise.
<i>nil</i>	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**

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

### **Value Returned**

<code>r_path</code>	<code>r_path</code> with the circle as the only segment.
<code>nil</code>	<code>axlPath</code> 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.
```

## **axlPathOffset**

```
axlDB2Path(  
    r_path  
    xy  
)  
==> r_path
```

### **Description**

Adds an offset, `xy`, to all points within a `r_path`.

### **Arguments**

<code>r_path</code>	
<code>offset</code>	<code>offset xy</code>

### **Value Returned**

`new r_path`

### **See Also**

[axlPathStart](#), [axlDB2Path](#)

### **Examples**

Obtain shape outline as a `r_path` and then move it by 10:20.

```
p = ashOne("shapes")  
path = axlDB2Path(p)  
path1 = axlPathOffset(path 10:20)
```

## axlDB2Path

```
axlDB2Path(  
    o_dbid  
)  
==> r_path
```

### Description

This takes a database id (`o_dbId`) and converts it to an `r_path`. This function supports all dbids with a segment attribute. For example, shape, void, path, and line.

**Note:** In AXL-SKILL terminology, path and line, refers to cline/line and segments, respectively.

### Arguments

`o_dbId`      The dbid for the line.

### Value Returned

- `r_path` - if object can be converted
- `nil` - object cannot be converted.

**See Also:** [axlPathStart](#)

### Examples

To obtain shape outline as a `r_path`, use following commands.

```
p = ashOne("shapes")  
path = axlDB2Path(p)
```

## ax1DBCreatePath

```
ax1DBCreatePath(  
    r_path  
    [t_layer]  
    [t_netName] / ['line]  
    [o_parent]  
    [lo_props]  
    [s_font]  
)  
⇒ l_result/nil
```

### Description

Creates a path figure (line or cline) as specified. Does not add a net name to etch when the etch is not connected to a pin, via, or shape. If etch is added, it ties to the first net it touches, otherwise it remains “not on a net” as specified by the arguments described below.

Clines may merge with other clines so that the resulting coordinates may be a superset of the provided coordinates. This is not currently true for line types.

Normally, if you want to attach properties to a newly-created object, call `ax1DBAddProp()` after creating the object. CLINES may merge with existing CLINES, so the object you end up adding properties to may not match the one you created. The `lo_props` option deals with this issue. You can add properties when you create the CLINE and if the property list on your CLINE differs from any merged target CLINES, your CLINE will not merge.

LINES with the interface are supported even though lines do not merge.

Allegro restricts the layers that allow fonts (`s_font`). ETCH layers may never have fonted lines.

## Arguments

<i>r_path</i>	Existing path consisting of the straight-line and arc segments previously created by <code>axlPath</code> 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 belongs. If the net <i>t_netName</i> does not exist, <code>axlDBCreatePath</code> does not create any path, and returns <code>nil</code> .
'line	By default a path created on an Etch layer creates clines unless you pass 'line for this argument.
<i>o_parent</i>	<i>dbid</i> of object to be the parent of the path figure. Use the symbol <code>instance</code> or use <code>nil</code> 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.
<i>[lo_props]</i>	Optional list of property name/value pairs. (See <code>axlDBAddProp()</code> for format.)
<i>[s_font]</i>	Optional line font, may have values as 'SOLID 'HIDDEN 'PHANTOM 'DOTTED 'CENTER. <code>nil</code> is the same as 'SOLID

## Value Returned

<i>l_result</i>	Returns a list: <ul style="list-style-type: none"><li>■ (car) list of <i>dbids</i> of all path figures created or modified</li><li>■ (cadr) <code>t</code> if DRCs are created. <code>nil</code> if DRCs are not created.</li></ul>
<code>nil</code>	Nothing was created.

## See Also

### [axlDBAddProp](#)

## Example

```
path = axlPathStart( list 100:0 100:500)  
  
; create path on current default layer
```

## **Allegro SKILL Reference**

### Database Create Functions

---

```
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")

;create a cline path on top etch layer with properties
propList = list( '(FIXED t) )
axlDBCreatePath(path "ETCH/TOP" "gnd" nil propList)
```

## axlDBCreateLine

```
axlDBCreateLine(  
    l_points  
    [f_width]  
    [t_layer]  
    [t_netname]/['line']  
    [rd_parent]  
    [s_font]  
)  
⇒ l_result/nil
```

### Description

Create a path of fixed width straight segments, a line with series of provided points. If line is on an ETCH layer a cline will be created unless overridden with the 'line symbol.

All points are absolute in user units. For <n> points provided, the function creates <n-1> segments.

Allegro restricts the layers that allow fonts. ETCH layers may never have fonted lines.



The t\_netname option is used as a tie breaker in cases where the cline may want to connect to multiple objects. A cline cannot maintain a net by itself. This connectivity behavioral means if adding multiple clines and vias you must sequence them so that each cline or via is added such that it connects to an object with the desired net already in the database.

For example, you add 3 clines to drive a connection between 2 pins; where cline 1 and 3 terminate on a pin and cline 2 is in the middle. You should add them as 1 2 then 3. Adding them as 2, 1 and 3 may result in cline 2 being connected to a difference net.

## Arguments

<i>l_points</i>	List of the vertices (at least two) for this path.
<i>f_width</i>	Width for all segments in the path. Default is 0.
<i>t_layer</i>	Layer to which to add the path. Default is the current active layer.
<i>t_netname</i>	Name of the net or <i>nil</i>
<i>rd_parent</i>	<i>dbid</i> of the object to which the line is added. Use the symbol instance <i>dbid</i> or use <i>nil</i> to specify the design itself.
<i>s_font</i>	Optional line font, may have values as 'SOLID, 'HIDDEN 'PHANTOM 'DOTTED 'CENTER. <i>nil</i> is the same as 'SOLID

## Value Returned

<i>l_result</i>	List: (car) list of <i>dbids</i> of all paths created or modified (cadr) <i>t</i> if DRCs are created. Otherwise the function returns <i>nil</i> .
<i>nil</i>	Nothing is created.

## See Also

[axIDBCreatePath](#)

## Example

```
axIDBCreateLine( (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.

## **axIDBCreateCircle**

```
axIDBCreateCircle(  
    l_location  
    [f_width]  
    [t_layer]  
    [rd_parent]  
)  
⇒ l_result/nil
```

### **Description**

Create a circle at indicated location and with indicated diameter.

### **Arguments**

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

### **Value Returned**

<i>l_result</i>	List containing:  (car) list of circle <i>dbids</i> . There is always one <i>dbid</i> in the list. (cadr) t if any DRCs are created. <i>nil</i> if no DRCs are created.
<i>nil</i>	Nothing was created.

### **See Also**

[axIDBCreatePath](#)

## 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))
```

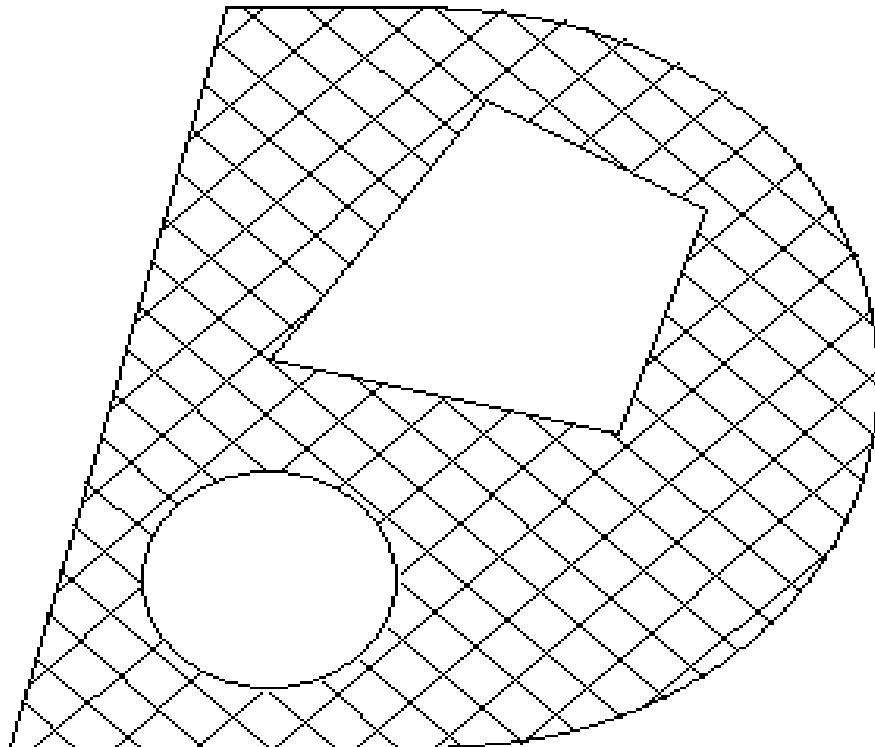
## Allegro SKILL Reference

### 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



## **ax1DBComposeShapesFromLines**

```
ax1DBComposeShapesFromLines (
    o_dbid/lo_dbids
    n_maxGap
    n.cornerRadius
    t_layer
    b_shapeFill
    [t_netName]
    [b_deleteLines]
)
⇒ o_dbid/lo_dbids/error string
```

### **Description**

Use this function to convert a group of lines and arcs created from a DXF or Gerber input into a shape. The function encapsulates the `compose` shape command. It allows you to gather the same objects and programmatically convert those into shapes on the target layers.

Any shape created from lines using this function is equivalent to the shape created from the `ax1DBCreateOpenShape` and `ax1DBCreateShape` functions. This function provides convenient access to features like bridging gaps in lines, which are brought in from external files. It can also round or trim corners the same way as the interactive compose shape tool does.

## Arguments

<i>o_dbid/lo_dbids</i>	Any combination of line segments, lines, and shapes in the database for conversion. The shapes may become voids in other shapes, and so on, based on the nesting of the resulting shapes that are created.
<i>n_maxGap</i>	The maximum air gap between vertices, which should be close enough to create a closed shape outline. This argument corresponds to the <i>Maximum gap</i> option of the <code>compose shape</code> command. If <code>nil</code> , the value is computed from the provided segments.
<i>n_cornerRadius</i>	Radius for corners, if corners are to be rounded. This argument corresponds to the <i>Radius</i> option of the <code>compose shape</code> command. If <code>nil</code> , corners are not rounded.
<i>t_layer</i>	Destination layer on which shape is to be created
<i>b_shapeFill</i>	<p>Either <code>t</code> or <code>nil</code>:</p> <ul style="list-style-type: none"><li>■ <code>t</code>: creates a solid-filled shape</li><li>■ <code>nil</code>: creates an unfilled shape</li></ul> <p>Certain layers are allowed to have only one type of shape. If the specified destination layer only allows one shape fill state, this argument is ignored, and the layer's required shape type is used.</p>
<i>[t_netName]</i>	Net assigned to the shape. Only applicable when creating shapes on cross-section layers.
<i>[b_deleteLines]</i>	<code>t/nil</code> indicates whether the lines used in the construction of the shapes should be deleted. Lines not incorporated into a shape are never deleted.

## Value Returned

`o_dbid/l_dbids/  
string`

One of three possible results may be returned:

- `o_dbid`: A single shape dbid, if only one shape (with or without voids) is created
- `lo_dbids`: List of all shape dbids, if multiple shapes are created from provided lines
- `string`: On error or invalid arguments, a string indicating the problem.

## Notes

- Some layers have restrictions on objects which are allowed. The `axlDBComposeShapesFromLines` function abides by these restrictions when creating shapes from the lines. For instance, unfilled shapes cannot be created in the `PLACE_BOUND_TOP/BOTTOM` layers.
- The `DESIGN_OUTLINE` layer is unique. This layer may only have a single (unfilled) shape outline on it. If multiple shapes result from the operation, or if shapes already exist on this layer, then all additional shapes are assumed to be cutouts in the design outline and are created on the `CUTOUT` layer, instead.
- Details about the process are available in the `axl_compose_shape_from_lines.log` file. This file is provided for debugging during development. The file will be overwritten by the next call to the API.

## Example

```
; Define path and create as a line in database  
path = axlPathStart( list(0:0 0:500 500:500 500:0 0:0))  
line = car(axlDBCreatePath(path "DRAWING FORMAT/OUTLINE" 'line))  
; Convert line to rectangular design outline on design outline layer.  
; Leave original path line on the DRAWING FORMAT layer for reference.  
outline = axlDBComposeShapesFromLines(line nil nil  
"BOARD GEOMETRY/DESIGN_OUTLINE" nil nil nil)
```

## **axlDBCreateBoundingShape**

```
axlDBCreateBoundingShape (
    lo_dbids
    t_padLayer
    n_expansion
    t_shapeLayer
    [t_netName]
    [b_deleteExisting]
)
⇒ list(o_dbid b_drcs)
```

### **Description**

Creates a convex hull, also called a bounding shape, around a set of objects. Objects supporting a convex hull are pins, vias, bond fingers, and cline segments. Other objects passed to the function are filtered from the *lo\_dbids*.

## Allegro SKILL Reference

### Database Create Functions

---

#### Arguments

<i>lo_dbids</i>	Cline segments, vias, pins, and bond fingers for which the function calculates and creates the convex hull. All other items are removed from the list. A minimum of two items is required to create a convex hull shape.
<i>t_padLayer</i>	Pad layer for vias, pins, and bond fingers. The same layer must be used for all pads. Cline segments may exist on any layer.  <b>Note:</b> The regular metal pad is used from the layer specified.
<i>n_expansion</i>	Expansion to apply around all objects when calculating convex hull.  Positive value provides additional clearance from the edge of pads, while negative value keeps the shape inside the edge of pads.
<i>t_shapeLayer</i>	Layer on which shape is to be created.
<i>[t_netName]</i>	Net assigned to the shape. Only applicable when creating shapes on cross-section layers.
<i>[b_deleteExisting ]</i>	<i>t/nil</i> . If true, any shape on the specified target layer overlaps the new bounding shape is removed.

#### Value Returned

<i>list(o_dbid b_drcs)</i>	dbid of shape and boolean indicator of whether DRCs have resulted.
<i>nil</i>	Error in parameters sent to function

#### Notes

Convex hulls represent the outline of a shape you would get by snapping a rubber band around the set of objects. It will not have inlets or concave sections in the boundary. It is a strictly positive shape outline.

#### Example

```
; Select all vias in the -500:-500 500:500 visible window.  
axlSetFindFilter(?onButtons list("NOALL" "VIAS") ?enabled list("NOALL"  
"VIAS"))  
axlAddSelectBox(list(-500:-500 500:500));
```

## **Allegro SKILL Reference**

### Database Create Functions

---

```
vias = axlGetSelSet();
; Create bounding shape around these vias on the SOLDERMASK_TOP layer, removing
any old
; soldermask shape found overlapping.
shape = axlDBCreateBoundingShape(vias "ETCH/TOP" 50.0 "BOARD GEOMETRY/
SOLDERMASK_TOP" nil t)
```

## axIDBCreateOpenShape

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

### Description

Creates a shape based on the characteristic of either *o\_polygon* or *r\_path*. With *r\_path*, fills 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*, creates a shape with the boundary defined by the boundary of the polygon. The holes in the polygon are added as voids to the shape. (See ax1PolyFromDB.)

The shape model uses the open/close model for performance reasons. While adding a shape without voids, you can use axIDBCreateShape, which hides the open and close. While adding voids you should do the following:

```
shape = axIDBCreateOpenShape(...)  
... add voids ...  
axIDBCreateCloseShape(shape).
```

You can modify an existing shape by using the axIDBOpenShape, as follows:

```
axIDBOpenShape(shape <new boundary>)  
... add or delete voids ...  
axIDBCreateCloseShape(shape).
```

Will not allow hole polygons as input. When holes are passed as input, the following warning displays:

```
Invalid polygon id argument -<argument>
```

See axIDBCreateSymDefSkeleton on notes about restrictions on shapes that are part of symbol definitions.

A static shape is created if you create shape on class ETCH; dynamic shapes are created if class is BOUNDARY. For example, to create a static shape on the TOP layer, make *t\_layer=ETCH/TOP*. To make a dynamic shape, make *t\_layer=BOUNDARY/TOP*. The same rule also applies to ax1DBCreateShape.

## Allegro SKILL Reference

### Database Create Functions

---

fill structure for xhatch shapes is:

l_fill1	A fill_type.
[l_fill2]	(optional) A fill_type. Supplied when more than second xhatch pattern is desired.
[f_outlineWidth]	(optional) Width of outline must be greater than or equal to fill width(s). Specified in design units. Default is current board xhatch width. Only supported for <i>o_polygon</i> since outline width for <i>r_path</i> should be supplied through the <i>r_path</i>

where fill\_type is a defstruct with members

f_spacing	spacing between xhatches (design units)
f_width	width of xhatch (design units)
l_origin	origin of xhatch (absolute to board)
f_angle	angle of xhatches

## Arguments

<i>o_polygon/r_path</i>	The outline as an <i>r_path</i> from axlPathXXX data structure or an <i>o_polygon</i> from axlPolyXXX interfaces.
<i>l_r_fill</i>	List of fill structures ( <i>r_fill</i> ) for non solid fill shapes or: <i>t</i> → solid fill <i>nil</i> → unfilled
<i>t_layer</i>	Layer name. <i>nil</i> uses the default active layer.
<i>t_netname</i>	Name of net. Only allowed for shapes being added to etch layers.
<i>o_netdbid</i>	Can use <i>DBID</i> of net instead of the netname. Same restrictions apply as for <i>t_netname</i> .
<i>o_parent</i>	axl <i>DBID</i> of the object to add the shape to. Use the symbol instance, or use <i>nil</i> to specify the design itself.

## Value Returned

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

## Notes

- 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.
- Only one shape can be in open state at one time.

## See Also

[axlDBActiveShape](#), [axlDBOpenShape](#), [axlDBCreateVoid](#), [axlShapeDeleteVoids](#),  
[axlDBCreateCloseShape](#), [axlDBCreateRectangle](#), [axlDBCreateShape](#),  
[axlDBCreateVoidCircle](#), [axlShapeAutoVoid](#), and [axlDBCreateFillet](#)

## Example

### ■ Create a shape using rpath

```
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))
```

### ■ Create a shape using a poly

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

### ■ See examples `axldbctshp.iil`.

## axlDBCreateCloseShape

```
axlDBCreateCloseShape (
  o_shape
  [g_forceShape]
)
⇒ l_result/nil
```

### Description

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

### Arguments

<i>o_shape</i>	<i>dbid</i> of the open shape created by axlDBCreateOpenShape.
<i>g_forceShape</i>	By default, Allegro creates a rectangle in its database when an outline is a rectangle. This is for performance and space reasons. To override this behavior, pass the argument value as t when closing a shape.
<i>r_fill</i>	Shape can be filled differently than voided. This should only be done with xhatch shapes and should use spacing/width in power of two multiples. We strongly discourage this option. Used in place of g_forceShape.

### Value Returned

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

### Example

See [Create Shape Interface](#) on page 1054 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 1054 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 1054 for an example.

## **axIDBCreateVoid**

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

### **Description**

Adds a void to a shape. To add multiple voids, it is recommended that you either add the voids when creating the shape ([axIDBCreateShape](#)) or re-open the shape ([axIDBOpenShape](#)) before creating the voids.

Only certain layers, such as ETCH layer, allow voids in a shape. Use [axIOK2Void](#) to determine if shape supports voids. While adding multiple voids to an etch shape, for best performance, first call [axIDBOpenShape](#), add the voids, and then close the shape ([axIDBCreateCloseShape](#)).

Unless you want the void to be permanent, do not add voids to dynamic shapes. User added voids on dynamic shapes must be put on the dynamic shape, with class=BOUNDARY, not on the generated shape, class=ETCH.

### **Arguments**

<i>o_shape</i>	<i>dbid</i> of the open shape. If this value is <i>nil</i> , the command uses the open shape
<i>r_path</i>	Existing path structure created by the <a href="#">ax1Path</a> 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.

### **See Also**

[axIDBCreateShape](#), [axIDBOpenShape](#), [axIOK2Void](#), [axIDBCreateVoidCircle](#)

### **Example**

- Create Shape Example

See [Create Shape Interface](#) on page 1054 for an example.

- Add to existing shape

See `dbc_shp_t10` function in `ax1dbctshp.i1` example code.

## **ax1DBCreateShape**

```
ax1DBCreateShape(  
    o_polygon/r_path  
    [l_r_fill]  
    [t_layer]  
    [t_netName]  
    [o_parent]  
)  
⇒ l_result/nil
```

### **Description**

Takes the same arguments as `ax1DBCreateOpenShape` 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 `ax1DBCreateShape` to add shapes without voids.

`ax1DBCreateShape` has the same argument restrictions as `ax1DBCreateOpenShape`.

## ArgumentsZ

<i>o_polygon/r_path</i>	Existing path structure created by <code>axlPath</code> functions.
<i>l_r_fill</i>	One of three possible values: <i>t</i> → create shape solid filled <i>nil</i> → create shape unfilled
	List of structures specifying crosshatch parameters for creating the shape: <pre>(defstruct axlFill: (<i>r_fill</i>) - shape crosshatch data   origin: a point anywhere on any xhatch line   width: width in floatnum user units   spacing: spacing in floatnum user units   angle): angle of the parallel lines)</pre>
	<b>Note:</b> As with all SKILL defstructs, use the constructor function <code>make_axlFill</code> to create instances of <code>axlFill</code> . Use the copy function <code>copy_axlFill</code> to copy instances of <code>axlFill</code> .
<i>t_layer</i>	Layer on which to create the shape.
<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 1054 for an example.

## **axIDBCreateRectangle**

```
axIDBCreateRectangle(  
    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 Allegro PCB Editor layer requiring an unfilled rectangle.
- Attempt to create an unfilled rectangle on an Allegro PCB Editor layer requiring a filled rectangle.

See [axIDBCreateSymbolSkeleton](#) for notes about restrictions on shapes that are part of symbol definitions.

## Allegro SKILL Reference

### Database Create Functions

---

#### Arguments

<i>l_bBox</i>	Bounding box of the rectangle: Lower left and upper right corners of rectangle
<i>g_fill</i>	If <i>t</i> then the fill is solid. If <i>nil</i> (default) then the rectangle is unfilled. It may have optional line font, with possible values as 'SOLID 'HIDDEN 'PHANTOM 'DOTTED 'CENTER. Line fonts can only be used with unfilled shapes. Only certain Allegro layers support fonted unfilled shapes.
<i>t_layer</i>	Layer to which to add the rectangle. Default is the active layer.
<i>t_netname</i>	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.
<i>o_parent</i>	<i>dbid</i> of object of which the rectangle is to be a part. Use either the <i>dbid</i> of a symbol instance or use <i>nil</i> to specify the design itself.

#### Value Returned

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

#### Example

- Unfilled shape indicated layer

```
axlDBCreateRectangle(list(100:100 200:200) nil "BOARD GEOMETRY/OUTLINE")
```

- Filled shape on active layer

```
axlDBCreateRectangle(list(200:200 400:300) t)
```

- Filled shape on ETCH/TOP assigned to NET\_1 using user supplied picks

```
axlDBCreateRectangle( axlEnterBox() t "ETCH/TOP" "GND")
```

- Fonted dotted line shape on indicated layer

```
axlDBCreateRectangle(list(400:100 500:300) 'DOTTED "BOARD GEOMETRY/OUTLINE" )
```

## **axIDBCreateVectorizedEllipse**

```
axIDBCreateVectorizedEllipse(
    l_center
    n_radiusA
    n_radiusB
    n_ccwRotationAngleOfEllipse
    n_numSegs
    [g_fill]
    [t_layer]
    [t_netname]
    [o_parent]
)
⇒ l_result/nil
```

### **Description**

Creates a vectorized ellipse at the specified location, with given parameters and number of ellipse segments. If the ellipse is not created, the function returns `nil`.

If `t_netname` is non-null, then the ellipse becomes a member of that net.

`axIDBCreateVectorizedEllipse` ignores `t_netname` if the ellipse is unfilled.

The function does not create the ellipse and returns `nil` (error) in the following instances:

- The net does not exist.
- Attempt to create a filled ellipse on an Allegro PCB Editor layer requiring an unfilled ellipse.
- Attempt to create an unfilled ellipse on an Allegro PCB Editor layer requiring a filled ellipse.

See [axIDBCreateSymbolSkeleton](#) on notes about restrictions on shapes that are part of symbol definitions.

## Arguments

<i>l_center</i>	The location of the center of the ellipse
<i>n_radiusA/</i> <i>d_radiusB</i>	Half of the axes of the ellipse
<i>n_ccwRotationAng1</i> <i>eOfEllipse</i>	Angle by which the ellipse is rotated counterclockwise
<i>n_numSegs</i>	Number of segments to use in creating the ellipse
<i>g_fill</i>	If <i>t</i> then fill is solid. If <i>nil</i> (default) then the ellipse is unfilled. It may have optional line font, with possible values as 'SOLID 'HIDDEN 'PHANTOM 'DOTTED 'CENTER. Line fonts can only be used with unfilled shapes. Only certain layout editors layers support fonted unfilled shapes.
<i>t_layer</i>	Layer to which to add the ellipse. Default is current active layer.
<i>t_netname</i>	Name of net of which this ellipse is to be a part. This argument is meaningful only if the ellipse is being added on an Etch layer.
<i>o_parent</i>	<i>dbid</i> of object of which the ellipse is to be a part. This can be either the <i>dbid</i> of a symbol instance or <i>nil</i> for the design itself.

## Value Returned

<i>l_result/nil</i>	<i>nil</i> if nothing was created. Otherwise it returns <i>l_result</i>
<i>list</i>	List: <ul style="list-style-type: none"><li>■ * (car) <i>ellipse dbid</i></li><li>■ * (cadr) <i>t</i> if DRCs created. Otherwise returns <i>nil</i></li></ul>

## Example

- Unfilled shape indicated layer

```
axlDBCreateVectorizedEllipse(100:100, 100, 55, 0, 22, nil "BOARD GEOMETRY/  
OUTLINE")
```

- Filled shape on active layer

```
axlDBCreateVectorizedEllipse(100:100, 100, 55, 0, 14, t)
```

## **Allegro SKILL Reference**

### Database Create Functions

---

- Filled shape on ETCH/TOP assigned to NET\_1 using user supplied picks

```
axlDBCreateVectorizedEllipse( 100:100 100.0 55.0 0.0 14 t "ETCH/TOP")
```

## Nonpath DBCreate Functions

This section describes the `DBCreatE` functions that add nonpath figures to the Allegro PCB Editor database.

### **axlCreateBondFinger**

```
axlCreateBondFinger(
    parentSymbol
    fingerName
    list(fingerLocation fingerRotation fingerPadstack)
    list(placementStyle ewlLength fingerSnap fingerAlign)
)
==> dbid/nil
```

#### **Description**

This function adds a valid, fully-instantiated bond finger to the database. Bond fingers created through this interface can be safely manipulated by the wirebond toolset and will also be properly recognized by all aspects of the database (DRC, signal integrity, 3D viewer, and so on).

#### **Arguments**

<code>parentSymbol</code>	dbid of the symbol (generally a die) with which this finger should be associated when performing operations like a move or delete.
<code>fingerName</code>	The optional parameter that specifies the name of the bond finger, as stored in the <code>BOND_PAD</code> property.
<code>fingerLocation</code> <code>Rotation</code> <code>Padstack</code>	The physical information about the bond finger being creation, the location is a database coordinate point, the rotation and angle in degrees, and the padstack the dbid of a padstack to use.
<code>placementStyle/ewlLength</code> <code>fingerSnap/fingerAlign</code>	The placement data for the bond finger being created, as follows:

## Allegro SKILL Reference

### Database Create Functions

---

placementStyle	String value in the following list: <ul style="list-style-type: none"><li>■ Orthogonal</li><li>■ Equal Wire Length</li><li>■ On Path</li><li>■ Free Placement</li></ul>
ewlLength	Length value for Equal Wire Length style, which represents the desired length of the wire.
fingerSnap	String value in the following list: <ul style="list-style-type: none"><li>■ Center of Finger</li><li>■ Finger Origin</li><li>■ Near End</li><li>■ Far End</li><li>■ Nearest Point</li><li>■ Farthest Point</li></ul>
fingerAlign	String value in the following list: <ul style="list-style-type: none"><li>■ Aligned with Wire</li><li>■ Orthogonal to Die Side</li><li>■ Orthogonal to Guide</li><li>■ Pivoting Ortho to Guide</li><li>■ Average Wire Angle</li><li>■ Constant Angle</li><li>■ Match CW Neighbor</li><li>■ Match CCW Neighbor</li></ul>

#### Value Returned

- `dbid` of newly created bond finger if successful.
- `nil` if an error occurred (message printed to status window).

## Allegro SKILL Reference

### Database Create Functions

---

## axlCreateBondWire

```
axlCreateBondWire(  
    parentSymbol  
    list(wireStartOwner wireStartLocation)  
    list(wireEndOwner wireEndLocation)  
    list(wireDiameter wireProfile)  
)  
==>dbidt/nil
```

### Description

This function adds a valid, fully-instantiated bond wire to the database. Bond wires created through this interface can be safely manipulated by the wirebond toolset and will also be properly recognized by all aspects of the database (DRC, signal integrity, 3D viewer, etc).

### Arguments

parentSymbol	dbid of the symbol (generally a die) with which this wire should be associated when performing operations like a move or delete.
wireStartOwner/Location	Optional.  This is a list with the first item being, the dbid of the object to which the start of the wire attaches. If this object is a pin or finger, the location will be derived from the object's origin. If the object is a shape, you must pass the location for the connection as well.
wireEndOwner/Location	- This is a list with the first item being, the dbid of the object to which the end of the wire attaches. If this object is a pin or finger, the location will be derived from the object's origin. If the object is a shape, you must pass the location for the connection as well.
wireDiameter/Profile	- This list of two items describes the physical placement of the wire in terms of its 3D profile (a string) and the wire diameter (a number).

## **Allegro SKILL Reference**

### Database Create Functions

---

#### **Value Returned**

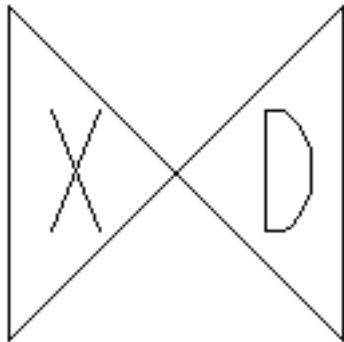
- dbid of newly created bond wire if successful.
- nil if an error occurred (message printed to status window).

## axlDBCreateExternalDRC

```
axlDBCreateExternalDRC(
    t_constraint/lt_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.



You may pass the constraint as the traditional argument (*t\_constraint*) where this contains both the constraint and expected value in a one string. The downside of this method is that the `show element` and `reports` commands report 0 for the expected value. Alternatively, you can pass it as a list containing two strings: constraint name and expected value. This format reports properly in both the `show element` and `reports` commands.

The *t\_actualValue* argument is optional and provides an externally-defined actual value with the DRC in design units.

In `show element`:

SKILL Item	Show Element	Comments
<code>l_anchorPoint</code>	Origin xy	Required Value
<code>t_constraint</code>	Constraint set	Required Value

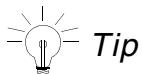
## Allegro SKILL Reference

### Database Create Functions

---

SKILL Item	Show Element	Comments
t_type	Constraint Type	default "EXTERNAL REFERENCE"
t_expectValue	Constraint value	default "None"
t_actualValue	Actual Value	default "None"
Property mapping		
t_expectValue	EXTERNAL_DRC_VALUE	

**Note:** Attempting to create a DRC object on a non-DRC class is an error. You can use this function in the layout editor, but not in the symbol editor.



*Tip*  
t\_type can be used to group similar DRCs. You might give all of your company written DRC checks your company name.

## Arguments

<i>t_constraint</i>	Name of the violated constraint. String contains the type of constraint and the required value and comparison.
<i>lt_constraint</i>	Alternative method. It is a list of ( <i>t_constraint t_expectValue</i> )
<i>l_anchor_point</i>	Coordinate of the DRC marker.
<i>t_layer</i>	Layer of the DRC marker. This must either include the DRC error class or just the subclass name.
<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) (cdr) <i>t</i> (always) (in future, <i>t</i> if DRC is new, else <i>nil</i> if DRC existed already)
<i>nil</i>	Nothing is created.

## Example

Creates a user-defined DRC marker at x,y (1500, 1800) to mark a violation of user rule: "Line to Pin--MY SPACING RULE" with a required value of 12 and an actual value of 10.

■ Original method:

```
ax1DBCreateExternalDRC( "MY Spacing Line to Pin/req:12" 1500:1800 "drc  
error calss/top", nil, nil, "10 MILS")
```

■ New method for better show element and reports command behavior:

```
ax1DBCreateExternalDRC('("My Spacing Line to Pin" "12")  
1500:1900 "top", nil nil "10 MILS")
```

■ Name classification

## **Allegro SKILL Reference**

### Database Create Functions

---

```
axlDBCreateExternalDRC(  
    ' ("My Spacing Line to Pin" "12" "Cadence")  
    1500:2000 "top", nil nil "10 MILS")
```

Adds "X D" DRC markers at (1500 1800) and (1500 1900).

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  
- - - - -
```

## **axIDBCreateFillet**

```
axIDBCreateFillet(  
    o_polygon/r_path  
    t_layer  
    o_filletObject  
)  
⇒ l_result/nil
```

### **Description**

Adds a fillet shape to the object.

See [axIDBCreateOpenShape](#) for description of the first argument.

The provided shape must touch the connect point of the parent object.

Tees cannot be directly selected. You must select a cline that it is connected and use [axIDBGetConnect](#) (basic mode) to find the Tee of interest. The cline may have a Tee on either end.

Typical reasons for failure to create a fillet are:

- The object must exist on the layer (in case of vias or pins it must span the object).
- It must touch or overlap the connect point of the object.
- The cline must not be a thermal.

### **Notes**

- To delete a fillet use [axIDeleteFillet](#).
- Fillets are automatically removed if the fillet object is modified.
- If adding multiple fillets use [axIDBCloak](#).
- Fillets created with this API will be reported as fillets throughout Allegro.
- Fillets may cause DRCs.
- The partial fillet type is not supported.
- Fillets cannot be added in symbol editor.
- Additional info on failures may be returned by setting [axIDebug \(t\)](#)

## Arguments

<i>o_polygon/r_path</i>	The outline as an r_path from axlPathXXX data structure or an o_polygon from axlPolyXXX interfaces.
<i>t_layer</i>	Layer name. Must be an etch layer and the parent object must exist on that layer.
<i>o_filletObject</i>	axl DBID of object to add the Fillet. Must be a Via, Pin or Tee.

## Value Returned

<i>l_result/nil</i>	nil if not created, or a list containing (car) axl DBID of the shape (cadr) t if DRCs created or nil.
---------------------	---

## See Also

[axlDBCreateOpenShape](#), [axlDBGetConnect](#), [axlDeleteFillet](#).

## **axlDBCreatePin**

```
axlDBCreatePin(  
    t_padstack/o_padstackDbid  
    l_anchorPoint  
    r_pinText/nil  
    [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.

### **Notes:**

- 1) This interface may only be used in the Symbol Editor.
- 2) Use axlDBCreatePin only in package and mechanical symbol drawings. Creating a pin in any other type of drawing causes errors.
- 3) Use *nil* for *r\_pinText* to create a mechanical pin.

### **Arguments**

<i>t_padstack</i>	Padstack name for the via. 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.
<i>o_padstackDbid</i>	a padstack dbid
<i>l_anchorPoint</i>	Layout coordinates of the location to add the pin.

## Allegro SKILL Reference

### Database Create Functions

---

*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) ;textOrientation - ;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
```

```
; 'GEOMETRY --> only geometry is mirrored
```

```
justify) ;"left", "center", "right"
```

**Note:** As with all SKILL defstructures, use constructor functions `make_axlPinText` to create instances of `axlPinText` and `make_axlTextOrientation` for `axlTextOrientation`.

See [Create Shape Interface](#) on page 1054 for an example. Use copy functions `copy_axlPinText` to copy instances of `axlPinText` and `copy_axlTextOrientation` for `axlTextOrientation`.

*f\_rotation* Rotation of pin in degrees.

## Value Returned

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

## Example

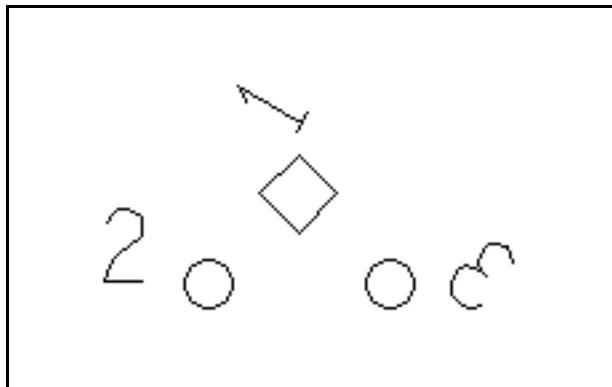
- The following example adds pins "1", "2", "3", and a mechanical to a package symbol drawing. Pin "1" with a square pad is rotated 45 degrees, pins "2" and "3" with round pads, and pin "3" with its pin text mirrored.

```
mytext = make_axlTextOrientation(  
                                ?textBlock 6, ?rotation 60.0  
                                ?mirrored nil ?justify "center")  
  
mypad = make_axlPinText(?number "1",  
                        ?offset 0:75, ?text mytext)  
axlDBCreatePin( "pad1" 0:0 mypad 45.0)  
mytext->justify = "left"  
mytext->rotation = 0.0  
mypad->number = 2  
mypad->offset = -125:0  
axlDBCreatePin( "pad0" -100:-100 mypad)  
mytext->rotation = -45.0  
mytext->justify = "right"  
mytext->mirror = t  
mypad->number = 3  
mypad->offset = 50:0  
axlDBCreatePin( "pad0" 100:-100 mypad)  
mypad->text = nil  
axlDBCreatePin( "pad0" 100:100 mypad)
```

## Allegro SKILL Reference

### Database Create Functions

Adds the three pins in the positions shown:



- 2) Create 8 pins using a loop

```
x=1000.0
```

```
y=1000.0
```

```
myText = make_axlTextOrientation(?textBlock 6  
                                ?justify "center")  
myPin= make_axlPinText(?offset 0:0 ?text myText)  
  
for(i 1 8  
    y=y-100  
    sprintf(buf "a%d" i)  
    myPin->number = buf  
    axlDBCreatePin("VIA" x:y myPin)  
)  
)
```

## **axlDBCreateSymbol**

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

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

### **Description**

Places a symbol instance in the design. 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:

<i>t_refdes</i>	Reference designator of the component. 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 <code>nil</code> if it cannot find the given refdes.
-----------------	---

## Allegro 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* nil - create unmirrored. (default)

t - create symbol mirrored.

'GEOMETRY - geometry is mirrored.

*f\_rotation* Rotation of the symbol in degrees.(default is 0)

## Allegro SKILL Reference

### Database Create Functions

*t\_embeddedLay* Place on embedded layer. Layer must be enabled for embedded.

*er* Mirror option is ignored. Layer may either be fully qualified

("ETCH/GND") or just the subclass ("GND"). May not use the top  
or bottom layer.

#### Value Returned

*nil* Nothing is created.

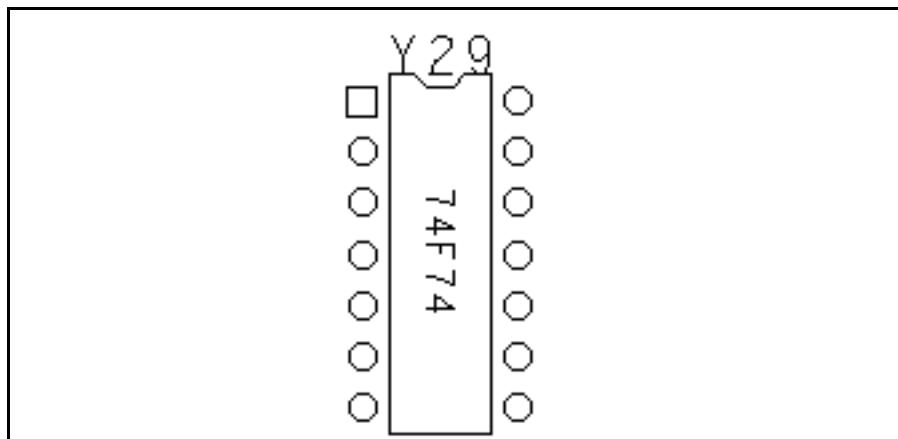
*l\_result* a list containing:  
(*car*) *axl DBID* of the symbol created  
(*cadr*) *t* if DRCs are created. *nil* if DRCs are not created.

**Note:** The symbol definition in the drawing is used. If there is none in the drawing, then the symbol library is searched and the definition loaded.

#### Example

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

Creates a symbol with the assigned refdes.

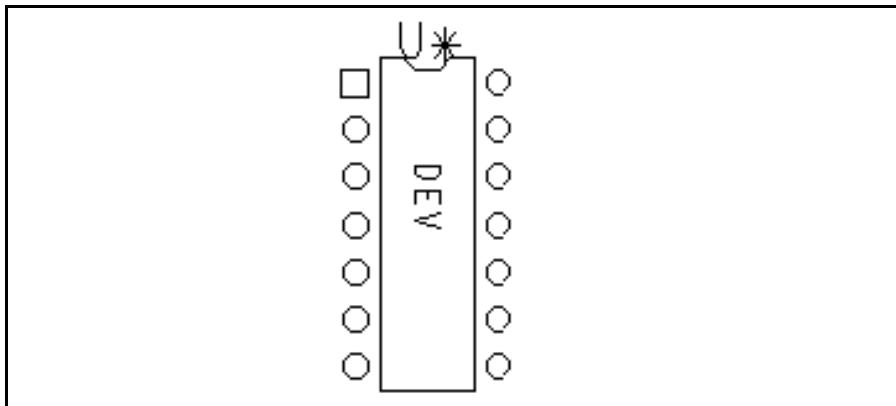


```
axlDBCreateSymbol( list( "dip14" "package"), 5600:4600)
⇒(dbid:423144 nil)
```

## Allegro SKILL Reference

### Database Create Functions

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



- Typical component driven symbol placement:

```
p = axlDBCreateSymbol("U1" 2175:1000)
```

- Place an embedded component. Assumes a layer, SIGNAL\_2 is enabled for embedded.

```
p = axlDBCreateSymbol("R1" 2175:1000 nil nil "SIGNAL_2")
```

- Place non-logical package symbol with rotation of 90

```
p = axlDBCreateSymbol('("R_0402" "PACKAGE") 2000:800.1 nil 90.0)
```

## axlDBCreateSymbolSkeleton

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

or

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

### Description

Places a skeleton or 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* defstructures 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 Allegro 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 Allegro PCB Editor database model. It provides a powerful method for creating symbols within Allegro PCB Editor. Although you can use this command to create non-conventional symbols, the rest of Allegro 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. Allegro 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 Allegro PCB Editor symbol.***

## Arguments

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

<code>t_refdes</code>	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 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 a refdes; if *t\_refdes* is present, *t\_symbolType* must be "package"

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

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. This will be the origin of the symbol.

*g\_mirror* nil - create unmirrored (default).

t - create symbol mirrored.

'GEOMETRY - geometry is mirrored.

*f\_rotation* Rotation angle of the symbol in degrees.

nil - 0.0.

## Allegro SKILL Reference

### Database Create Functions

---

*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.

*t\_embeddedLayer* Place on embedded layer. Layer must be enabled for embedded. Mirror option is ignored. Layer may either be fully qualified ("ETCH/GND") or just the subclass ("GND"). May not use the top or bottom layer.

### Value Returned

*l\_result* nil – Nothing is created.  
List Containing the following:  
(car) *axl DBID* of the symbol created  
(cadr) *t* if DRCs are created. *nil* if DRCs are not created.

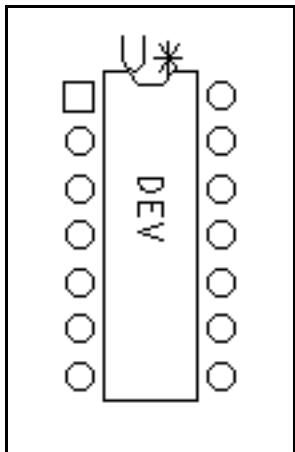
### Example

```
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)
```

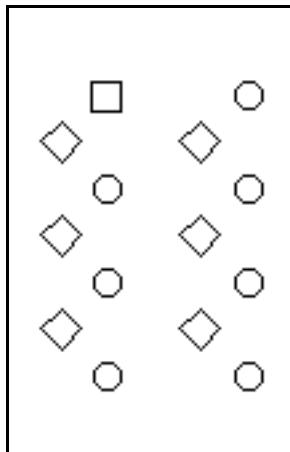
## Allegro SKILL Reference

### Database Create Functions

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**

*t\_text* 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 *dbids* of all text lines it creates. The *textBlock* parameter block specified in the *axlTextOrientation* structure specifies spacing between multiple text lines.

*l\_anchorPoint* Layout coordinates of the location to add the text.

## Allegro SKILL Reference

### Database Create Functions

---

*r\_textOrientation* a *axlTextOrientation* structure:

```
tion          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,
              'GEOMETRY --> only geometry is mirrored
              justify) ;"left", "center", "right"
```

**Note:** As with all SKILL defstructs, use the constructor function `make_axlTextOrientation` to create instances of `axlTextOrientation`. Use the copy function `copy_axlTextOrientation` to copy instances of `axlTextOrientation`.

*t\_layer* Name of the layer on which the text is to be added.

*o\_attach* DBID of the object to which the text must be attached, or use `nil` for the design.

### Value Returned

<i>l_result</i>	Otherwise the function returns a list: (car) list of text <i>DBIDs</i> created, one for each line of text input (cadr) t if DRCs are created. Otherwise the function returns nil.
<code>nil</code>	Nothing is created.

### Notes:

- If *o\_attach* is a symbol instance, then the text is “stand alone”, but a child of the symbol instance.
- If the *t\_text* string contains NEWLINEs, then multiple text records will be created (and multiple *DBIDs* returned).

## See Also

[axlTextOrientationCopy](#), [axlDBChangeText](#)

## Example

The following example adds the e text string “Chamfer both sides” center justified, mirrored and rotated 60 degrees.

```
myorient = make_axlTextOrientation(?textBlock "8", ?rotation 60.0,  
                                    ?mirrored t, ?justify "center")  
  
ret = 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/o_padstackDbid  
    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 for the via. 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.
<i>o_padstackDbid</i>	A padstack dbid
<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; <i>nil</i> → via is stand-alone.
<i>g_mirror</i>	<i>t</i> → create mirrored via. <i>nil</i> → create unmirrored via. 'GEOMETRY → only via geometry is mirrored. Via layers are not mirrored 'LAYERS → only via layers are mirrored. Via geometry is not mirrored.
<i>f_rotation</i>	Rotation angle of via in degrees.
<i>o_parent</i>	axl DBID of the object to which to attach the via. Use a symbol instance or use <i>nil</i> to specify the design itself.

## Value Returned

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

## Note:

axlDBCreateVia cannot create a test point. You have to create testpoints by using the axlTestPoint function.

## **Allegro SKILL Reference**

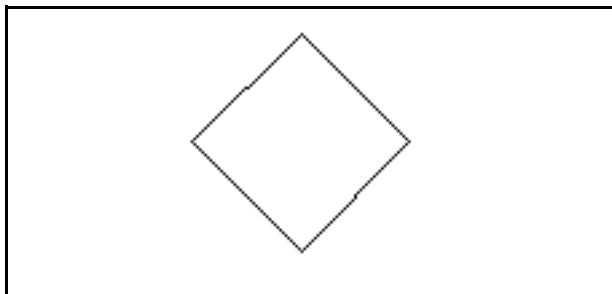
### Database Create Functions

---

#### **Example**

```
myvia = axlDBCreateVia( "pad1", 5600:4200,  
    "sclk1", t, 45.0, 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:



## **axIDBCreateSymbolAutosilk**

```
axIDBCreateSymbolAutosilk (
    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.

## **axlDBCreateViaStructure**

```
axlDBCreateViaStructure(
    t_refdes/o_vsDbid
    l_anchorPoint
    [t_netName/o_netDbid]
    [g_mirror]
    [f_rotation]
    [g_snapFlag]
)
==> l_result/nil
```

### **Description**

Verify arguments and add the structure instance with specified parameters in the design. Check out the Allegro database id.

### **Arguments**

<i>t_refdes</i>	Structure symbol name
<i>o_vsDbid</i>	Structure symbol reference
<i>t_netName</i>	Return path net name
<i>o_netDbid</i>	Return path net reference
<i>l_anchorPoint</i>	The location of the structure
<i>g_mirror</i>	<ul style="list-style-type: none"><li>■ nil: not (default)</li><li>■ t: Structure is mirrored</li></ul>
<i>f_rotation</i>	The rotation angle in degrees (default is 0 degree)
<i>g_snapToLocation</i>	Specifies if structure needs to be snapped to an object at location assuming it shares common layer (snaps to pin, via, dangling cline). In case of snapping structure inherits connectivity from snapped object. <ul style="list-style-type: none"><li>■ nil: Structure is not snapped (default)</li><li>■ t: Structure is snapped</li></ul>

## Value Returned

l_result	List: (car) list of DBID of all symbols created (cadr) t if DRCs are created or nil
nil	if not created

## Examples

- Typical structure placement:

```
p = axlDBCreateViaStructure("VS1" 2175:1000)
```

- Place high-speed via structure with return path

```
p = axlDBCreateViaStructure("VS2" 2175:1000 "Vss")
```

- Place standard via structure with rotation on 90 and snapping

```
p = axlDBCreateViaStructure("VS3" 2000:800 nil nil 90.0 t)
```

## **axlCreateWirebondGuide**

```
axlCreateWirebondGuide(  
    r_path  
)  
==> dbid/nil
```

### **Description**

This function adds a wirebond guide path into the design, which can then be used to snap fingers through the wirebond tools.

### **Arguments**

<code>r_path</code>	Existing path consisting of the straight-line and arc segments previously created by <code>axlPath</code> functions
---------------------	---

### **Value Returned**

- `dbid` of newly created guide path if successful.
- `nil` if an error occurred (message printed to status window).

## **axlZoneCreate**

```
axlZoneCreate(  
    t_name  
    o_dbidOutline  
) ==> o_zoneDbid/nil
```

### **Description**

Creates a zone with the name *t\_name* and with *o\_dbidOutline*. Should use the axlZoneAccess API to trim the zone outline and associated shapes to the design and zone outlines. The behavior of Allegro PCB Editor is undefined if zone shapes overlap the design outline or other zone shapes.

Recommended steps to create a zone:

- Create zone outline shape on RIGID\_FLEX/ZONE\_OUTLINE class and subclass
- Create a zone using [axlZoneCreate](#)
- Set the characteristics including the associated stackup using [axlZoneAccess](#)
- Trim the zone shapes using [axlZoneAccess](#)

Side effects:

- DRC is marked out of date
- Creates attached text with zone name
- Outline cannot be modified or deleted
- Zone allows full placement

### **Arguments**

<i>t_name</i>	Name of zone
<i>o_dbidOutline</i>	Outline (shape) on class==RIGID_FLEX and subclass==ZONE_OUTLINE

### Value Returned

*t*                    If successful

nil                  If failed

### See Also

[axlZoneSet](#), [axlZoneDelete](#), [axlZoneAccess](#)

`axlDBGetDesign() ->zone` returns a list of all zones of a design.

### Example

Create a zone (assumes a shape (dbid) exists) on class==RIGID\_FLEX and subclass==ZONE\_OUTLINE

```
axlDBCreateZone ("MY_ZONE" shape)
```

## 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/t
    [ln_range]
    [t_units]
    [g_hidden])
)
⇒ od_propDictEntry/nil

axlDBCreatePropDictEntry(
    nil
)
==> lt_availbeObject
```

#### Description

Creates an Allegro user-defined property dictionary entry with given attributes. Once a dictionary entry is created, the property can then be attached to objects.

STRING property values are limited to 1024. STRING\_ID allows property values up to 4096. STRING\_ID is not currently supported in "define property" dialog of Allegro PCB Editor.

If you need to store larger data within the database, use attachments ([axlCreateAttachment](#)).

## **Allegro SKILL Reference**

### Database Create Functions

---

#### **Arguments**

*t\_name* Name of the property. Must be different from all other property names in the design, both Allegro PCB Editor pre-defined and user-defined property names.

*t\_type* Data type of the property value.

Legal values are:

Typical: BOOLEAN, INTEGER, REAL, STRING, and DESIGN\_UNITS.

Other supported types are:

ALTITUDE

CAPACITANCE

DISTANCE

ELEC\_CONDUCTIVITY

FAILURE\_RATE

IMPEDANCE

INDUCTANCE

LAYER\_THICKNESS

NAME

NOISE\_VOLTAGE

PERCENTAGE

PROP\_DELAY

RESISTANCE

TEMPERATURE

THERM\_CONDUCTANCE

THERM\_CONDUCTIVITY

THERM\_RESISTANCE

VOLTAGE

VELOCITY

STRING\_ID

## Allegro SKILL Reference

### Database Create Functions

---

<i>lt_objects</i>	List of strings representing the object types to which this property can be added. (Use <code>axlDBGetPropDictEntry</code> ( <code>nil</code> ) to get a list of valid objects). If only a single object type is allowed, then it may be specified as a string, rather than a list containing one string.  If this value is <code>t</code> then all allowed properties are allowed.
<i>ln_range</i>	List of the lowest and highest legal values for the (numeric) property. If the first value is <code>nil</code> , it means negative infinitely. If the second value is <code>nil</code> , it means infinity.
<i>t_units</i>	A text string so be used with data types ( <i>t_type</i> ) without units, such as STRING, INTEGER, or REAL.
<i>g_hidden</i>	<i>t</i> property is hidden from the user. Hidden properties are not shown in any Allegro UI like Constraint Manager, Show Element or Property Edit. Hidden properties can be accessed via SKILL. Typically, properties are hidden if they are only meant to be changed outside of the SKILL program. Hidden properties are also visible via <code>extracta</code> .

#### Value Returned

<i>o_propDictEntry</i>	<i>DBID</i> of the property dictionary entry created.
<code>nil</code>	Property not created.

#### See Also

[axlDBAddProp](#), [axlCreateAttachment](#)

#### Example

- Add a new property of type string, supported on db objects

```
propDoct = axlDBCreatePropDictEntry("ACME" "STRING" t)
```
- Create MYPROP as a real number property with range -50 to 100 units of "level", attachable to pins, nets, and symbols.

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

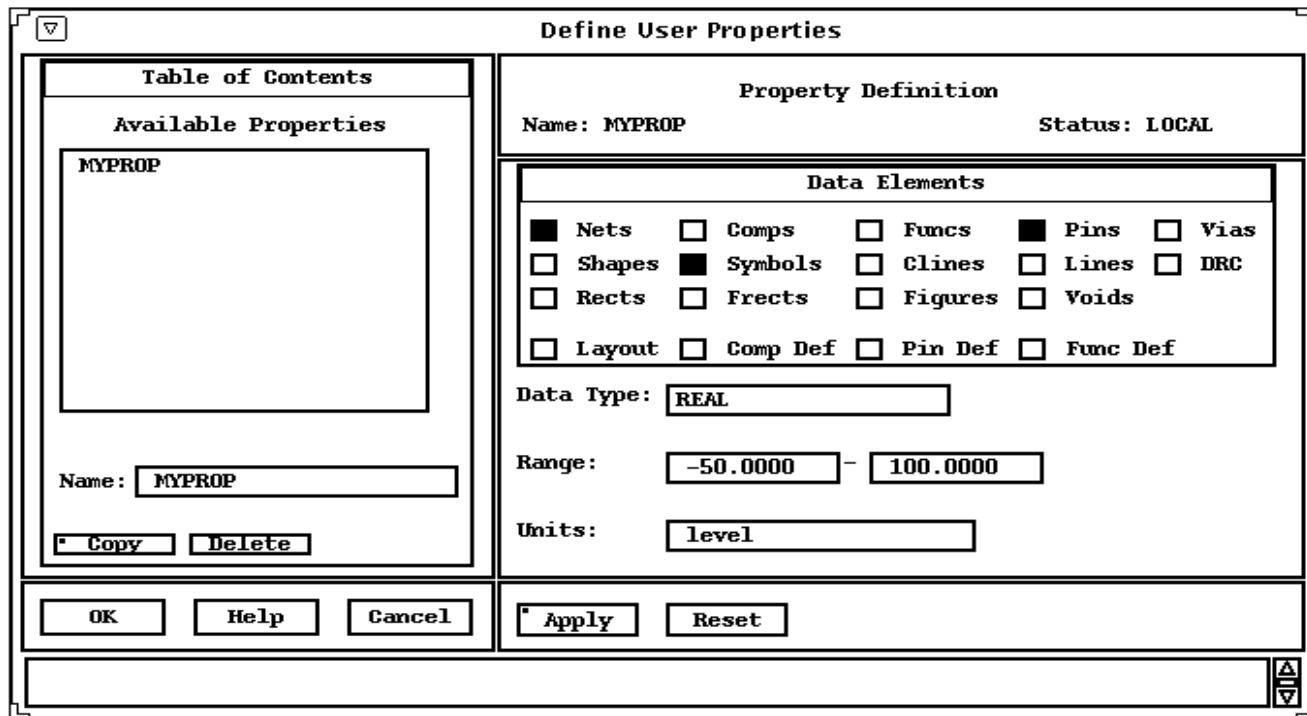
## Allegro SKILL Reference

### Database Create Functions

To check

1. From Allegro 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 the property/value pairs listed in *ll\_name\_value* to the object database DBIDs listed in *lo\_attach*. If an object does not accept a property name in *ll\_name\_value*, ax1DBAddProp ignores that combination, and continues. If an object already has the specific property attached, ax1DBAddProp 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

*lo\_attach*

List of Allegro PCB Editor object dbids to which the property/value combinations listed in *ll\_name\_value* is to be added. A list of *nil* denotes attachment to the design (*list nil*). However, if *lo\_attach* is *nil*, there are no objects for attachment, and `axlDBAddProp` does nothing, returning *nil*.

*ll\_name\_value*

List of property-name/property-value pairs. If the first element of the list is not a list, `axlDBAddProp` treats *ll\_name\_value* as a single name-value list and assigns the value to the property of that name.

The first element of each name-value pair is the property name as a string. The second element of the name-value list is the property value. The property value is either a string with or without units included, or a simple value (fixed or floating number). In either case, if units are not included explicitly, `axlDBAddProp` uses the units specified in the system `units.dat` file.

A BOOLEAN property has no value, the property either exists on the object or it does not. `axlDBAddProp` ignores the value for a BOOLEAN property and simply adds the property to the object. If you query the value of a BOOLEAN property it will return `t` if the property exists. You can call `axlDBAddProp` for a BOOLEAN property with the name of the property; for example, `(list "fixed")`.



*Caution*

***Calling `axlDBAddProp` for a BOOLEAN property and passing the value `nil` will still add the property to the object. For example,***

```
axlDBAddProp('dbid) '(fixed" nil))
```

***will add the fixed property to the object. If you query the value of "fixed" on that object later, it will return a value of t to indicate that the property exists, even though the originally specified value was nil. To make a BOOLEAN property query return nil, use***

```
axlDBDeleteProp to remove the property from the object. STRING type properties allow the value to be set to t or nil, even though those are not string values. t is an alias for the string "TRUE", while nil is an alias for the string "FALSE". For example, axlDBAddProp((list nil) (list "COMMENT" t)) will add a COMMENT property to the design object with the value "TRUE". Trying to set any other property type to t or nil will produce an error.
```



***Even though STRING properties support setting the value to "TRUE" or "FALSE", Cadence strongly recommends against using a STRING property to represent BOOLEAN values. STRING type properties are slower to access and test than BOOLEAN properties and require more storage in the database. Therefore, using STRING properties to store BOOLEAN values can lead to performance problems.***

## Value Returned

l_result	List:  (first element) list of <i>dbids</i> of objects with at least one property successfully added  (second element) always nil.
nil	No properties are added.

## See Also

[axIDBDeleteProp](#), [axIDBCreatePropDictEntry](#) [axIDBGetPropDictEntry](#)  
[axIDBGetProperties](#), [axIDBDeletePropAll](#), [axIDBDeletePropDictEntry](#), and  
[axIDBGetPropDict](#)

## Example

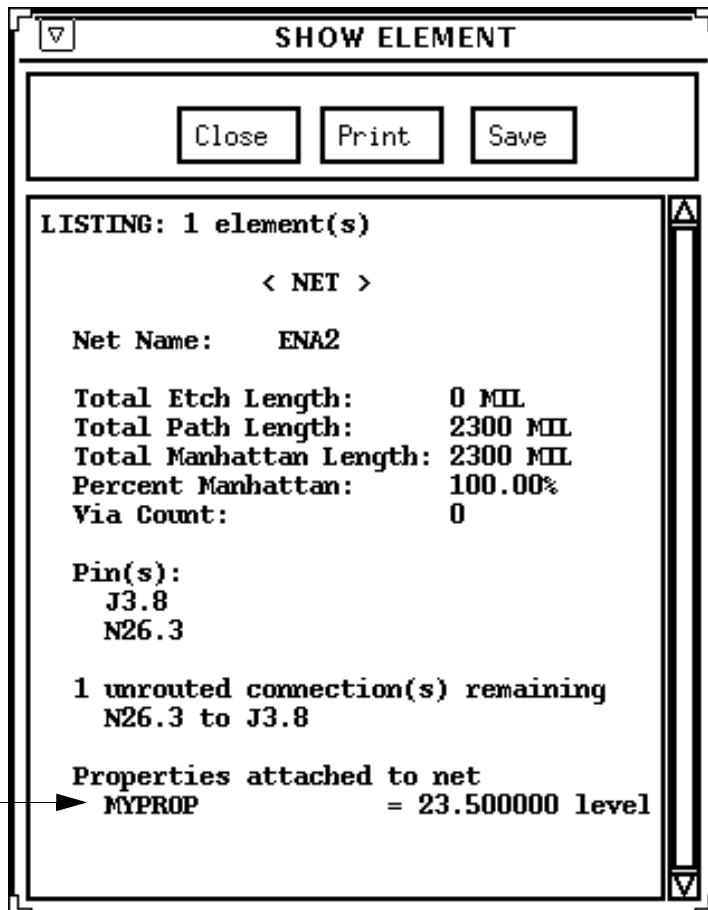
see [axIDBDeleteProp](#)

## Allegro 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 and Save Functions

This section describes the *Load* functions that add external objects to the Allegro 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, Allegro PCB Editor looks in the pad library on the disk.

#### **Arguments**

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

#### **Value Returned**

<i>o_dbid</i>	<i>dbid</i> of padstack loaded.
<i>nil</i>	Nothing is found.

#### **Example**

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

Loads the VIA padstack.

## axlLoadSymbol

```
axlLoadSymbol(  
    t_symKind  
    t_symName  
) -> o_dbidSymDef/nil
```

### Description

Searches for indicated symbol in database. If not present, searches PSMPATH and loads the symbol into the database. In the symbol editor, this can only be used for shape and mechanical symbols for use with padstacks.



***If a symbol definition is not in use (dbid->instance is nil) then the definition is deleted. This deletion of unused symbols occurs during save drawing, refresh symbol, place manual among other places. This means the database is saved as part of axlRunBatchDBProgram then the unused symdefs will be deleted.***

FLASH and SHAPE symbols are loaded automatically when a padstack using those symbols is loaded. This interface allows loading of these symbol types to allow analysis of the contents of these symbols types since you cannot use the extracta program.

### Notes:

- axlDBCreateSymbol also loads the symbol definition if required. You do not need this API to place symbols.
- You can delete unused symdefs via axlDeleteObject.

## Arguments

<i>t_symkind</i>	"PACKAGE", "MECHANICAL", "FORMAT" , "SHAPE, or "FLASH" (case insensitive)
<i>t_symName</i>	Name of symbol (lower case). This is the root name of the symbol, do not include an extension (for example, .psm) or a directory path.

## Value Returned

<i>dbid</i>	Of symbol definition
<i>nill</i>	Cannot find symbol, unknown symbol type, symbol type doesn't match symbol, can't find a padstack that is required for a sym pin, or symbol revision is too old.

## See Also

[axlDBCreateSymbol](#)

## EXAMPLE

```
symdef = axlLoadSymbol("package" "dip14")
```

## **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**

- Dump all the padstacks in the layout.

```
axlPadstackToDisk()
```

- Dump padstack "pad60cir36d" from the layout as "pad60cir36d.pad".

```
axlPadstackToDisk("pad60cir36d")
```

- Dump padstack "pad60cir36d" from the layout as "mypadstack.pad".

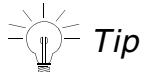
```
axlPadstackToDisk("pad60cir36d" "mypadstack")
```

## axlRefreshSymbol

```
axlRefreshSymbol(  
    t_symName/o_SymDef  
    [g_options]  
)  
==> t/nil
```

### Description

Refreshes a symbol from file on disk which is located by current PSMPATH. Works the same as the refresh\_symbol functionality except updates one symbol definition. Unlike refresh\_symbol this does not support the reset custom drill option since this is done at the padstack level not the symbol level.



- Tip*
- 1) If updating multiple symbols use [axlDBCloak](#) for best performance and minimal memory use.
  - 2) To ignore the FIXED property see [axlDBIgnoreFixed](#).

### Arguments

<i>t_symName</i>	existing symbol name
<i>o_SymDef</i>	symbol definition dbid
<i>g_options</i>	The available options are: <ul style="list-style-type: none"><li>■ 'text – reset text locations</li><li>■ 'fanout – reset fanouts (if design has fanouts delete them). Default is to only delete fanouts if disk symbol has them.</li><li>■ 'keepPadstack – keep instance edited padstacks</li></ul>

### RETURNS

- *o\_SymDef* – refreshed symdef
- *nil* – fails; typically if cannot find symbol on disk or if a FIXED property is present.

## Example

1. Update the DIP14 and reset text locations

```
axlRefreshSymbol("DIP14" 'text)
```

2. Update first symbol def off database root and set both the text and pin escape option

```
symdef = axlDBGetDesign()->symdefs  
axlRefreshSymbol(car(symdef) '(text fanout))
```

## See Also

[axlDBCloak](#), [axlDBIgnoreFixed](#), [axlReplacePadstack](#)

## **Allegro SKILL Reference**

### Database Create Functions

---

---

# **Database Group Functions**

---

## **Overview**

This chapter describes the AXL-SKILL Database Group functions.

## **ax1DBAddGroupObjects**

```
ax1DBAddGroupObjects (
  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 [ax1DBCreateGroup](#) 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 <a href="#">ax1DBCreateGroup</a> .

### **See Also**

[ax1DBCreateGroup](#)

## **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 are:

- "generic" - Allegro user groups
- members: anything except other non-generic group objects
- "bus"
- members: net and xnets
- "net\_group"
- members: xnet, net, diffpair, bus and net\_group
- "nets\_rko" - a net keepout group
- members: xnet, net, diffpair bus, net\_group shape and fill rectangles (via and route keepout)
- "die\_stack" - APD+ only
- members: compinst, syminst, rectangle, shape
- "ratbundle"
- members: pinpair
- "wire\_profile" - APD+ only
- members: a wire (cline or wire subclass)
- "module" - suggest to use `axlDBCreateModuleDef`

Accessible but not recommended.

- "db\_drill\_legend" - drill legend tables
- "rf\_nets" - RF "super" nets

## Allegro SKILL Reference

### Database Group Functions

---

*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*.

- For each group type has only certain objects that are allowed. For example generic groups only permits:

- group
  - component
  - symbol
  - net
  - path
  - via
  - shape
  - polygon
  - pin
  - text

- A circular group relationship cannot be formed.

For example, group A cannot 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.

*nil* If the group could not be created.

#### See Also

[axlDBAddGroupObjects](#), [axlDBDisbandGroup](#), [axlDBRemoveGroupObjects](#)

## Example

### ■ Generic group

```
groupMembers = axlGetSelSet()  
group_dbid = axlDBCreateGroup("my_group" "generic" groupMembers)
```

### ■ Net group

```
groupMembers = axlSelectByName("NET" "NET*" t)  
group_dbid = axlDBCreateGroup("NG1" "net_group" 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 <i>dbid</i> not being for a valid group.

### **See Also**

[axlDBCreateGroup](#)

## **axlDBGetGroupFromItem**

```
axlDBGetGroupFromItem(
  o_dbid
  t_groupType
  [g_promoteToNet]
) -> lo_groupDbid/nil
```

### **Description**

Filter object's group membership by a group type. You can normally fetch the list of groups that an object is a member of by using the groups attribute of its dbid (etc. o\_dbid->groups). This function provides additional filtering where you can request if an belongs to a particular group type. Depending upon the group characteristics an object can either belong to single group of a type or can be a member multiple groups of a single type. For example, an object can belong to multiple generic groups can belong to only one differential pair group.

The g\_promoteToNet option for groups for which membership is limited to nets/xnets. It promotes the object provided to its owning xnet, and is targeted for use with diffpair and bus groups, where membership is limited to the net's xnet. It provides an easy way for those groups if given the dbid of a net to promote the id to its xnet.

### **Arguments**

o_dbid	dbid to be examined
t_groupType	group type name. This is the group type name NOT the name of the group. In dbid terms this is group->type.
g_mode	If this value is t, it promotes the object to its net/xnet and performs test on that object.  If the value is set to 'all' then it traverses group hierarchy and returns both direct and indirect group membership of dbid.
	For example, if net is a member of a xnet which is a member of a diffpair, and you specify group type of "DIFF_PAIR", this returns diffpair group for object.

### Value Returned

lo\_groupDbid      Group dbids or nil if not a member of requested group type

**See Also:** [axlDBCreateGroup](#)

### Examples

In both case ashOne is a shareware utility that allows user to select one object (see  
`<CDSROOT>/share/pcb/examples/skill/ash-fxf/ashone.il`

- differential pair; set g\_promoteToNet to t in case net is part of a xnet

```
p = ashOne()      ; select a net that is a diffpair member  
l = axlDBGetGroupFromItem(p "DIFF_PAIR" t)
```

- generic group

```
p = ashOne()      ; create a group and select an object that is part of group  
l = axlDBGetGroupFromItem(p "GENERIC")
```

## **axIDBGroupRename**

```
axIDBGroupRename (
    o_groupDbid
    t_newName
)
==> t/nil
```

### **Description**

Renames a group. Groups supported are GENERIC, BUS, DIFF\_PAIR, NETCLASS, NET\_GROUP and MATCH\_GROUP. Do not attempt to rename group types not listed.

**Note:** All restrictions and disclaimers specified in [axIDBCreateGroup](#) also apply for this API.

### **Arguments**

<i>o_groupDbid</i>	The dbid of the group to be renamed
<i>t_newName</i>	New name of the group. Group name must be unique for the group type.

### **Value Returned**

<i>t</i>	Successful in rename.
<i>nil</i>	Failed in rename; dbid not a group, group can't be renamed, new name is not legal for group type or name already exists in that group type.

### **See Also**

[axIDBCreateGroup](#)

## **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 that 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.

### **See Also**

[axlDBCreateGroup](#)

## **axlNetClassAdd**

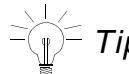
```
axlNetClassAdd(  
    o_netclassdbid/t_netclassName  
    o_dbid/lo_dbid  
) ==> t/nil
```

### **Description**

Adds members to a netclass group. Eligible members are:

- nets
- XNnets
- differential pairs
- busses

See netclass discussion in [axlNetClassCreate](#). This will mark DRC out of date. It is up to the application to update the DRC system.



*Tip*

Using dbids is faster then using names.

## Arguments

<i>o_netclassdbid</i>	dbid of a netclass group
<i>t_netclassName</i>	name of a netclass group
<i>o_dbid</i>	legal database dbid to add to netclass
<i>lo_dbid</i>	list of legal database dbids to add to netclass

## Value Returned

<i>t</i>	added elements
<i>n1l</i>	failed one or more element adds; object might already be a member of a netclass in that domain or not legal dbid to add to a netclass

## Examples

To netclass group created in axlNetClassCreate add two nets

```
nc = car(axlSelectByName ("NETCLASS" "5_MIL"))nets = axlSelectByName ("NET"  
' ("NET8" "NET9")  
axlNetClassAdd(nc nets)
```

## See Also

[axlNetClassCreate](#)

## **axlNetClassCreate**

```
axlNetClassCreate(  
    t_name  
    g_domain/lg_domain  
) ==> o_dbid
```

### **Description**

This creates a new netclass group. If a netclass exists with this name then `nil` is returned. Net Classes need to be populated via `axlNetClassAdd`. Empty net classes may be deleted on database save. A netclass must be part of one or more domains. These domains are shown below. The Same Net Constraint domain uses the netclass spacing domain. An object (bus, diffpair, xnet or net) may be a member of single netclass in a domain. For example, if net VCC exists in the POWER netclass in the physical domain then you cannot add it to another netclass in the physical domain. You can still add this net of a netclass in the spacing or electrical domain. You can obtain the current set of net classes in the database via: `axlDBGetDesign() ->netclass`. `axlNetClassGet` reports if an object is a member of a netclass either directly or via the logic hierarchy.

To assign a cset to a netclass assign the `PHYSICAL_CONSTRAINT_SET`, `SPACING_CONSTRAINT_SET`, `SAME_NET_SPACING_CONSTRAINT_SET` or `ELECTRICAL_CONSTRAINT_SET` property to the netclass where the value of the property is the cset name.

Same Net constraints shares the same domain with the `SPACING_CONSTRAINT_SET`.

## Arguments

<i>t_name</i>	name of netclass group (changed to upper case)
<i>g_domain</i>	netclass domain can be 'spacing', 'physical', 'electrical or 'all
<i>lg_domain</i>	list of netclass domains

## Value Returns

<i>nil</i> :	error or netclass with that name exists
<i>o_dbid</i> :	dbid of group

## See Also

[axlNetClassDelete](#), [axlNetClassAdd](#), [axlNetClassRemove](#), [axlNetClassGet](#), [axlDBAddProp](#), [axlCNSCreate](#)

## Examples

Create a netclass in physical domain called "5\_MIL"

```
nc = axlNetClassCreate("5_mil" 'physical)
```

## **axlNetClassDelete**

```
axlNetClassDelete(  
    o_netclassdbid/t_netclassName/lg_netclassdbid  
) -> t/nil
```

### **Description**

This deletes a net class group. It does not delete the objects belonging to the group. It is up to the application code to update DRC.

**Note:** Using dbids is faster then using names.

### **Arguments**

<i>o_netclassdbid</i>	dbid of a net class group
<i>t_netclassName</i>	name of a net class group
<i>lg_netclassdbid</i>	list of net class groups (dbids or names)

### **Value Returned**

<i>t</i>	net class group deleted
<i>nil</i>	failed

### **See Also**

[axlNetClassCreate](#)

### **Examples**

Delete net class group created in axlNetClassCreate

```
nc = car(axlSelectByName ("NETCLASS" "5_MIL"))  
axlNetClassDelete(nc)
```

or

```
axlNetClassDelete("5_MIL")
```

## **axlNetClassGet**

```
axlNetClassGet(  
    o_dbid  
    s_domain  
    g_hierarchal  
)==> o_netclass
```

### **Description**

Given a dbid (net, xnet, diffpair or bus) and a domain (spacing, physical or electrical) return its netclass. If g\_hierarchical is nil, returns object's netclass if a direct member. If g\_hierarchical=t returns first netclass encountered in logical hierarchy. For example, if a net is a member of a bus and the bus is assigned to netclass, BUSCLASS, and you pass a net of the bus to this API:

will return nil if g\_hierarchy=nil

will return netclass dbid, BUSCLASS, if g\_hierarchy=t

### **Arguments**

<i>o_dbid</i>	dbid may be net, xnet, diffpair or bus
<i>s_domain</i>	netclass domain; spacing, physical or electrical
<i>g_hierarchal</i>	

### **Value Returned**

<i>o_netclass</i>	dbid of netclass
<i>nil</i>	object not part of a netclass in the domain or an invalid object

### **See Also**

[axlNetClassCreate](#)

### **Examples**

Use example in [axlNetClassAdd](#)

From example in (should return netclass in both cases)

## **Allegro SKILL Reference**

### Database Group Functions

---

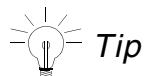
```
net = car(axlSelectByName("NET" "NET8"))
axlNetClassGet(nets 'physical nil)
axlNetClassGet(nets 'physical t)
```

## axlNetClassRemove

```
axlNetClassRemove(  
    o_netclassdbid/t_netclassName  
    o_dbid/lo_dbid  
) ==> t/nil
```

### Description

Removes elements from an existing net class group. Element must currently be a direct member of the group. This will mark DRC out of date. It is up to the application to update the DRC system.



*Using dbids is faster then using names.*

### Arguments

<i>o_netclassdbid</i>	dbid of a netclass group
<i>t_netclassName</i>	name of a netclass group
<i>o_dbid</i>	legal database dbid to remove from group
<i>lo_dbid</i>	list of legal database dbids to remove from group

### Value Returned

<i>t</i>	removed elements
<i>nil</i>	failed to remove one or more elements. Object may not be a cset member (member must be a direct member).

### Examples

Using the example from axlNetClassAdd remove one of the nets:

```
axlNetClassRemove (nc_car(nets))
```

### See Also

[axlNetClassCreate](#)

## axlRegionAdd

```
axlRegionAdd(  
    o_regiondbid/t_regionName  
    o_dbid/lo_dbid  
) ==> t/nil
```

### Description

Adds members to a region group. Eligible members are:

- shapes
- rectangles

Only objects on the CONS\_REGION class may be added to a region. See region discussion in axlRegionCreate. This will mark DRC out of date. It is up to the application to update the DRC system

**Note:** Using dbids is faster then using names.

### Arguments

<i>o_regiondbid</i>	dbid of a region group
<i>t_regionName</i>	name of a region group
<i>o_dbid</i>	legal database dbid to add to region
<i>lo_dbid</i>	list of legal database dbids to add to region

### Value Returned

<i>t</i>	added elements
<i>nil</i>	failed one or more element adds; object might already be a member of a region or not legal dbid to add to a region

### See Also

[axlRegionAdd](#)

## Examples

To region group created in axlRegionCreate add a shape

```
nc = car(axlSelectByName ("REGION" "BGA"))
lyr = "CONSTRAINT REGION/OUTER_LAYERS"
shape = axlDBCreateRectangle( list(100:100 200:200) nil lyr)
shape = car(shape)
axlRegionAdd(nc shape)
```

## **axlRegionCreate**

```
axlRegionCreate( t_name ) ==> o dbid
```

## Description

Creates a new region group. If a region exists with this name then `nil` is returned. Regions may contain shapes on `CONS_REGION` class. Shapes are added to the region group via the `axlRegionAdd`. Empty regions may be deleted on database save. You can obtain the current set of regions in the database via: `axlDBGetDesign ()->region`. None of the region APIs are enabled in the PCB L product.

**Note:** For better performance, when modifying regions you may wish to wrap all the calls with the axIDBCloak command.

## Arguments

*t\_name* name of region group (changed to upper case)

## Value Returned

*nil*: error or region with that name exists

*o\_dbid:* dbid of group

If shapes are a member of a region then their dbid region attribute will refer to the region dbid

## Examples

Create a region called "BGA"

```
nc = ax1RegionCreate ("BGA")
```

#### **See Also**

[axlRegionDelete](#), [axlRegionAdd](#), [axlRegionRemove](#), [axIDBCreateShape](#),  
[axIDBCreateRectangle](#)

## **axlRegionDelete**

```
axlRegionDelete(  
    o_regiondbid/t_regionName/lg_regiondbid  
) -> t/nil
```

### **Description**

This deletes a region group. It does not delete the objects belonging to the group.

**Note:** Using dbids is faster then using names.

### **Arguments**

<i>o_regiondbid</i>	dbid of a region group
<i>t_regionName</i>	name of a region group
<i>lg_regiondbid</i>	list of region groups (dbids or names)

### **Value Returned**

<i>t</i>	net class group deleted
<i>nil</i>	failed

### **See Also**

[axlRegionCreate](#)

### **Examples**

Delete region group created in axlRegionCreate

```
nc = car(axlSelectByName ("REGION" "BGA"))  
axlRegionDelete(nc)
```

or

```
axlRegionDelete("BGA")
```

## **axlRegionRemove**

```
axlRegionRemove(  
    o_regiondbid/t_regionName  
    o_dbid/lo_dbid  
) ==> t/nil
```

### **Description**

Removes shapes from an existing region group. Element must currently be a direct member of the group. This will mark DRC out of date. It is up to the application to update the DRC system.

**Note:** Using dbids is faster then using names.

### **Arguments**

<i>o_regiondbid</i>	dbid of a region group
<i>t_regionName</i>	name of a region group
<i>o_dbid</i>	legal database shapes to remove from group
<i>lo_dbid</i>	list of legal database shapes to remove from group

### **Value Returned**

<i>t</i>	removed elements
<i>nil</i>	failed to remove one or more elements. Object may not be a region member (member must be a direct member).

### **See Also**

[axlRegionCreate](#)

### **Examples**

Using the example from [axlRegionAdd](#) remove the shape:

```
axlRegionRemove(region shape)
```

**Allegro SKILL Reference**  
Database Group Functions

---

## **Allegro SKILL Reference**

### Database Group Functions

---

---

# **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 Allegro PCB Editor database attachment with the given attachment id. The attachment may optionally be given a password and a revision number. The 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.



#### **Caution**

***Do not create or replace attachments you do not own. This includes any predefined Allegro attachments like DFA or quickviews.***

## 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 <code>nil</code> .
<i>x_revision</i>	Revision number of the attachment. If <code>nil</code> , the revision number is set to zero.
<i>s_dataFormat</i>	Indicates the format of the <i>t_data</i> argument.  Permitted values:  'string — the value of the <i>t_data</i> argument is used for the attachment data.  'file — <i>t_data</i> references an ASCII file that is read into the database attachment.  'binary — <i>t_data</i> references a binary file that is read into the database attachment. Use this option with zip or compressed files.
<i>t_data</i>	

## Value Returned

<i>o_attachment</i>	AXL id for the new attachment, which can then be queried using the right arrow (->) operator.
<code>nil</code>	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.

## See Also

[axlGetAttachment](#)

## EXAMPLE

This uses an attachment to store in the database a list of variables. For example, you design a form where the user enters in their preferences and you manage them in SKILL via a disembodied property list. You would like to store the user's preferences with the design.

## Allegro SKILL Reference

### Database Attachment Functions

---

Create an attachment name. Create a name using an underscore, company name and application to make it unique. For example, \_acme\_bom\_rpt, would be a good attachment name.

```
attachName = "fxf"
```

; A typical disembodied property list

```
mylist = ncons(nil)
mylist->ccw = t
mylist->middle = nil
mylist->cx = 0.12
mylist->cy = 10.192
mylist->layer = "TOP"
```

**Note:** Do not store dbid's in the disembodied list or make sure to remove them before storing as an attachment.

Store list in current design (assuming user saves design)

```
dataString = sprintf(nil " '%L" mylist)
axlCreateAttachment(attachName nil 0 'string dataString)
```

Next time user runs the SKILL code, here is how to init the list:

```
attach = axlGetAttachment(attachName 'string)
if( attach)  then
    mylist = car(errsetstring(attach->data))
else      ; no list stored in design so init to default settings
    mylist = ncons(nil)
    mylist->ccw = t
)
```

## **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.



***Do NOT delete attachments you do not own. This includes any predefined Allegro attachments like DFA or quickviews.***

### **Arguments**

<i>t_attachmentId</i>	Id or name of the attachment to delete.
<i>t_passwd</i>	Password for this attachment.

### **Value Returned**

<i>t</i>	Attachment successfully deleted.
<i>nil</i>	Attachment not deleted.

### **See Also**

[axlGetAttachment](#)

## **axl GetAllAttachmentNames**

```
axlGetAllAttachmentNames (
    )
⇒ l_attachment/nil
```

### **Description**

Returns a list of the ids for all database attachments in the current Allegro PCB Editor database. If no attachments are present, then `nil` is returned. The attachments can retrieved using the [axlGetAttachment\(\)](#) function.

### **Arguments**

none

### **Value Returned**

<i>l_attachment</i>	List of attachment ids.
<code>nil</code>	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>'file</i> is displayed if you use ' <i>binary</i> option to create the attachment.
<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.



**Access to attachments in the private database is allowed. Do not create, change, or delete these attachments. The rule for attachments access is: If your application did NOT create the attachment do NOT change it.**

## 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 "data" attribute. Must be: <ul style="list-style-type: none"><li>■ 'string': return data as a string.</li><li>■ 'file': return data stored in a tmp file. You should use this option if you used 'binary' method to create the attachment.</li><li>■ nil: use the method describe in the attachment dataFormat attribute.</li></ul>

## 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
```

## See Also

[axlIsAttachment](#), [axlGetAllAttachmentNames](#), [axlCreateAttachment](#), [axlSetAttachment](#), [axlDeleteAttachment](#)

## **axlIsAttachment**

```
axlIsAttachment(  
    o_attachment  
)  
⇒ t/nil
```

### **Description**

Determines if the given object is an AXL attachment.

### **Arguments**

*o\_attachment*      Object to check.

### **Value Returned**

*t*      Object is an attachment.

*nil*      Object is not an attachment.

### **See Also**

[axlGetAttachment](#)

## axlSetAttachment

```
axlSetAttachment(  
    o_attachment  
    [t_password]  
)  
⇒ o_attachment/nil
```

### Description

Modifies an existing Allegro 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 `axl GetAllAttachments` 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

<code>o_attachment</code>	AXL id of the modified attachment.
<code>nil</code>	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 SKILL Reference**

### Database Attachment Functions

---

#### **See Also**

[axlGetAttachment](#)

**Allegro SKILL Reference**  
Database Attachment Functions

---

---

# Database Transaction Functions

---

This chapter describes the AXL-SKILL Database Transaction functions.

## axlDBCloak

```
axlDBCloak(
    g_func
    [s_mode]/[ls_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 *s\_mode* is 'shape'.
- Optionally, ignores the FIXED property if *s\_mode* is 'ignoreFixed'.
- Optionally, delays canvas redraws if *s\_mode* is 'disableDisplay'.
- Incorporate an errset around your function so any SKILL errors thrown are caught by axlDBCloak.

This function must be used if you need to update many etch or package symbols in a batch fashion.

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`.

gmode options (if multiple required pass a list of options

<code>'shape</code>	Improves performance if changes being made effect any dynamic shapes on the design. Generally you should set this if effecting ETCH layers with your changes.
<code>'ignoreFixed</code>	Have the system ignore the FIXED property (see <code>axlDBIgnoreFixed</code> )
<code>'disableDisplay</code>	Do not update the main canvas display during processing.

## CAUTIONS:

- A frequent programming error is to leave off the tick ('') mark that allows `axlDBCloak` to evaluate the function.

### CORRECT MODEL:

```
axlDBCloak( 'MyFunc(myArgs) ' (shape ignoreFixed))
```

### INCORRECT MODEL:

```
axlDBCloak( MyFunc(myArgs) ' (shape ignoreFixed))
```

Both work but in the incorrect case now performance benefit offered by the cloaking function will not be applied to `MyFunc`.

- Database transactions and cloaking (`axlDBTransactionStart`):
  - Do NOT start a transaction inside a cloak and then terminate it outside the cloak. Terminate means calling either `axlDBTransactionRollback`, `axlDBTransactionCommit` or `axlDBTransactionOops`.
  - If you start a transaction inside a cloak then complete it before returning.

- If you start a transaction outside the cloak then its completion must outside the cloak.



*Tip*

For effective debugging, first call your function directly from the top level function, then wrap in the cloak call.

## Arguments

<i>g_func</i>	Function with any of its arguments.
<i>s_mode</i>	option
<i>ls_mode</i>	list of options

## 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.

## **Allegro SKILL Reference**

### Database Transaction Functions

---

#### **See Also**

[axIDBTransactionStart](#)

## **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**

<i>x_mark</i>	Database transaction mark returned from <code>axlDBTransactionStart</code> .
---------------	---

### **Value Returned**

<i>t</i>	Mark written in the database.
<i>nil</i>	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 *oops* action.

### **Arguments**

*x\_mark*                    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 *oops*.

### **Example**

See `axlDBTransactionStart` for an example.

## **ax1DBTransactionRollback**

```
ax1DBTransactionRollback(  
    x_mark  
)  
⇒ t/nil
```

### **Description**

Undo function for a database transaction.

### **Arguments**

*x\_mark* Database transaction mark returned from ax1DBTransactionStart.

### **Value Returned**

t	Transaction undo completed.
nil	Transaction undo not completed.

### **Example**

See ax1DBTransactionStart for an example.

## axlDBTransactionStart

```
axlDBTransactionStart(  
    [g_undoMark])  
⇒ 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 `axlDBTransaction` 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

*g\_undoMark*      This should be set to 'undoMark' that are for commands that want multiple undo events. For example, when placing multiple individual symbols, you may want each placement to be an individual undo event.

By default, with a undo interactive all operations within the command results in a single undo even. See in `axlCmdRegister` on how write a SKILL command that supports Allegro undo.

### Value Returned

*x\_mark*      Integer mark indicating transaction start.  
*nil*      Failed to mark transaction start.

## Example 1

Emulates the Allegro *oops* model.

```
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
```

## Example 2

Multiple Start marks

```
i = axlDBTransactionStart()
... do stuff ...
j = axlDBTransactionStart()
... stuff ...
axlDBTransactionCommit(j) ;; this is not really committed
                           ;;till the outer commit
j = axlDBTransactionStart()
... do more stuff ...
axlDBTransactionRollback(j) ; oops out "do more stuff"

axlDBTransactionCommit(i) ;; commit changes to database
```

**Note:** Database transaction functions do NOT mark select sets. The application handles select set management.

## See Also

[axlDBTransactionMark](#), [axlDBTransactionOops](#), [axlDBTransactionCommit](#),  
[axlDBTransactionRollback](#), [axlCmdRegister](#)

**Allegro SKILL Reference**  
Database Transaction Functions

---

**Allegro 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. Via is added to end of list (see [axlCnsGetViaList](#) of via ordering functionality in etch editing).

Padstack does not need to exist to be added to a constraint via list.

If *t\_csetName* is *nil*, add padstack to all physical cssets.

**Note:** If a via already exists in the via list, a *t* is returned. Locked cssets return a *nil*.

### Arguments

<i>t_csetName</i>	Name of physical cset or <i>nil</i> for all cssets.
<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 cssets

```
axlCnsAddVia(nil "ALLPAD")
```

Add ONEPAD to DEFAULT cset

```
axlCnsAddVia("DEFAULT" "ONEPAD")
```

## **axICnsAssignPurge – Obsolete**

This function is obsolete. Kept for backward compatibility

## **axlCnsClassTableChange**

```
axlCnsClassTableChange (
    o_dbidClassTable
    s_csetType/l1_typeAndName
    [t_csetName]
) -> o_dbidClassTable/nil
```

### **Description**

This command changes the Csets associated with an existing net class table entry.

**Note:** See [axlCnsClassTableCreate](#) for complete family of functions.

You cannot change any table entries containing a region entry in Allegro PCB Designer or lower tiers.

### **Arguments**

<i>o_dbidClassTable</i>	dbid of an existing classTable entry.
<i>s_csetType</i>	symbol for cset type. spacing, 'physical or 'sameNet)
<i>l1_typeAndName</i>	lists <i>s_csetType</i> and <i>t_csetName</i>
<i>t_csetName</i>	cset name string

### **Value Returned**

Updated classTable dbid or nil if failure

### **See Also**

[axlCnsClassTableCreate](#)

### **Examples**

Before running the sample SKILLcode for `axlCnsClassTableChange`, create a class table by running the example code in the [axlCnsClassTableCreate](#) command. Next, execute the following SKILL code and update the entry created in [axlCnsClassTableCreate](#) by adding it to the spacing.

```
cset named 1, physical cset named 2 and a same net cset named 3.  
prop = '((spacing "1") (physical "2") (sameNet "3"))'
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

```
tbl = axlCnsClassTableChange(tbl prop)
```

## axlCnsClassTableCreate

```
axlCnsClassTableCreate(
    g_class1
    g_class2
    g_region
    s_csetType/11_typeAndName
    [t_csetName]
) -> o_dbidClassTable/nil
```

### Description

This command creates a class table entry that consists of any of the following:

- class to class (spacing only)
- region to class (spacing, same net and physical)
- region to class to class (spacing only)

Optionally, the command also associates a spacing cset, a physical cset, and same net cset with the table entry. If a class table entry already exists, it is modified with the provided csets.

Regions are not available in Allegro PCB Designer and lower products. Command will fail if you attempt to create a region-based table entry in these products. Class tables may not be created in symbol editor.

Points to remember:

1. The order of class1 and class2 does not matter.
2. If an entry already exists it will return the existing entry.
3. Net classes can be classified by domain (spacing and/or physical). If a netclass is restricted to one domain, it is possible to create a netclass to any entry that crosses domains. This table entry will be ignored by DRC. For example, you have a netclass ANY, in both physical and spacing domains; and another netclass PHYS that is restricted to the physical domain. It is possible to create a ANY to PHYS relationship which is only appropriate in the spacing domain but the PHYS netclass is not legal in that domain.

**Note:** This condition might be tested for and rejected in future releases.

4. DRC is set out-of-date, you must manually update the DRC.
5. Unlike Constraint Manager, you can add cset names that don't yet exist in the database. In these cases, we will automatically create a cset. Check via axlCnsList if your cset exists if you don't wish to create new csets.

6. Class table entries may also have constraint overrides attached via property overrides  
(axlDBAddProp)

## Arguments

<i>g_class1</i>	NETCLASS dbid or name of name class
<i>g_class2</i>	NETCLASS dbid, name of name class or nil
<i>g_region</i>	REGION dbid, name of region or nil
<i>s_csetType</i>	symbol cset type one of 'spacing, 'physical or 'samenet
<i>t_csetName</i>	string cset name for given type
<i>ll_typeAndName</i>	option list of values where you have  (( <i>s_csetType</i> <i>t_csetName</i> ) ( <i>s_csetType</i> <i>t_csetName</i> ) ...)

## Value Returned

returns dbid of type classTable for new or existing cset or nil if error

## See Also

[axlCnsClassTableFind](#), [axlCnsClassTableSeek](#), [axlCnsClassTableChange](#),  
[axlCnsClassTableDelete](#), [axlCnsList](#),

Also see classTable dbid object description.

## Examples

Create appropriate entries in design

```
region = axlRegionCreate("ANALOG")
ncls = axlNetClassCreate("VOLTAGE" '(spacing physical))
```

1. Add new spacing region-class table entry and give it the spacing cset "25MILS"

```
tbl = axlCnsClassTableCreate("VOLTAGE" nil "ANALOG" 'spacing "25MILS")
```

2. Alternative method plus also add a physical voltage cset.

```
props = '((spacing "25MILS") (physical "VOLTAGE"))
tbl = axlCnsClassTableCreate(ncls nil region props)
```

## **axlCnsClassTableDelete**

```
axlCnsClassTableDelete(  
    o_dbidClassTable/lo_dbidClassTable  
) -> t/nil
```

### **Description**

Deletes one or more entries in the class table.

DRC is marked out of date.

### **Arguments**

*o\_dbidClassTable* dbid of an existing classTable entry.

*lo\_dbidClassTable* deletes list of classTable entries.

### **Value Returned**

*t* if successful, *nil* if an error

### **See Also**

[axlCnsClassTableCreate](#), [axlCnsDeleteRegionClassClassObjects](#),  
[axlCnsDeleteClassClassObjects](#)

### **Examples**

Create a classTable entry by executing example code in [axlCnsClassTableCreate](#). Delete entry just created by running the following command.

```
axlCnsClassTableDelete(tbl)
```

## **axICnsClassTableFind**

```
axlCnsClassTableFind(  
    s_type  
    [o_dbid]  
) -> lo_dbidClassTable/nil
```

## Description

This command searches the class table for class table entries matching the search criteria.

## Arguments

*s\_type*      Specifies the type of search to perform. The options available are:

- `netclass – returns all class entries (all entries except for wire and component)
  - `classClass – returns all class to class entries
  - `classRegion – returns all class to region entries
  - `classClassRegion – returns all class to class to region entries
  - `wireProf – returns all wire profile entries (APD+ only)
  - `component – returns all component entries (APD+ only)
  - `match – returns all entries that contain provided region or class dbid (o dbid)

*o\_dbid* only applicable for the match option will return all class table entries containing the dbid.

## Value Returned

List of class table dbids matching search criteria or nil if no match is found.

## See Also

[axlCnsClassTableCreate](#), [axlCnsClassTableSeek](#), [axlSelectByName](#)

## Examples

1. Return all class entries that effect physical, spacing or same net DRC

```
tbl = axlCnsClassTableFind('netclass')
```

2. Return all entries that contain Region “ANALOG” entry (assumes design has a region called “ANALOG”)

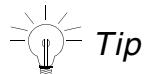
```
region = car (axlSelectByName("REGION" "ANALOG"))
tbl = axlCnsClassTableFind('match region')
```

## axlCnsClassTableSeek

```
axlCnsClassTableSeek(  
    g_class1  
    g_class2  
    g_region  
) -> o_dbidClassTable/nil
```

### Description

This command seeks a specific class table entry matching exactly the provided dbids. Order of class1 and class2 does not matter since C1/C2 is the same as C2/C1 and only one entry exists in the table.



*Tip*  
Constraint overrides may exist on a table entry via the `prop` attribute. While fetching multiple table entries, best performance is achieved by using dbids or using [axlCnsClassTableFind](#).

### Arguments

<code>g_class1</code>	NETCLASS dbid, name of net class or nil
<code>g_class2</code>	NETCLASS dbid, name of net class or nil
<code>g_region</code>	REGION dbid, name of region or nil

### Value Returned

Class table entry matching search criteria or `nil` if none found.

### See Also

[axlCnsClassTableCreate](#)

### Examples

Create a classTable entry by executing example code in [axlCnsClassTableCreate](#)

1. Return class table entry for a netclass called VOLTAGE and region called ANALOG.

```
tbl = axlCnsClassTableSeek("VOLTAGE" nil "ANALOG")
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **2. Alternative method using dbids**

```
region = car( axlSelectByName("REGION" "ANALOG"))
netclass = car( axlSelectByName("NETCLASS" "VOLTAGE"))
tbl = axlCnsClassTableSeek(netclass nil region)
```

## **axlCNSCreate**

```
axlCNSCreate(  
    g_domain  
    t_name  
    t_copyName  
)  
==> t/nil
```

### **Description**

Creates a new constraint set in the specified domain. For spacing and physical csets, you must supply an existing cset as the copy cset. If the *copyName* is nil, the DEFAULT cset is used. Electrical csets (ECsets) are created empty for a nil *copyName*. By default, the ECset created is empty. If you provide a second argument, the ECset contents are copied.

To assign a cset to a logical object such as a net, bus, or a netclass, assign a PHYSICAL\_CONSTRAINT\_SET, SPACING\_CONSTRAINT\_SET, SAME\_NET\_SPACING\_CONSTRAINT\_SET or ELECTRICAL\_CONSTRAINT\_SET property to the logical object where the value of the property is the cset name.

**Note:** Electrical csets cannot be created in Allegro PCB Designer.

## Arguments

<i>g_domain</i>	Specifies the domain of the Cset. Possible values are Physical, spacing, electrical, or 'sameNet'.
<i>t_name</i>	Name of new cset. (Changed to upper case) string must pass allowed character set
<i>t_copyName</i>	Name of cset to use as template. If this is <i>nil</i> , spacing and physical domains use DEFAULT as the template, while in case of electrical domain, an empty ECset is created.

## Value Returned

<i>t</i>	Cset created.
<i>nil</i>	Failed for the following reasons: domain name is illegal; name of cset is illegal; cset already exists; or <i>copyName</i> cset does not exist.

## See Also

[axlCNSEcsetCreate](#), [axlCNSDelete](#), [axlCnsList](#), [axlDBAddProp](#), [axlNetClassCreate](#)

## Example

Create a new physical cset called *foo*.

```
axlCNSCreate('physical "foo" nil')
```

## **axlCNSCsetLock**

```
axlCNSCsetLock(  
    g_domain  
    t_csetName  
    g_mode  
)  
==> t/nil
```

### **Description**

This locks or unlocks a constraint set in the given domain. See discussion in [axlCNSIsLockedDomain](#).

You should usually lock or unlock the entire domain since this matches the DRC user model. We provide this interface to temporary unlock a locked cset, make changes then reapply the lock.

**Note:** Changing the lock on a cset can take a considerable amount of time since DRC needs to be updated. In the spacing domain, dynamic shapes need to be updated. If doing other changes, consider cloaking axlDBCloak the entire process. This API already uses cloaking for the individual cset.

### **Arguments**

<i>g_domain</i>	domain of cset; 'physical, 'spacing, 'sameNet, 'electrical
<i>t_csetName</i>	cset name
<i>g_mode</i>	may either be <i>t</i> (to lock) or <i>nil</i> to unlock

### **Value Returned**

Returns *t* if updated lock status, *nil* an error.

### **Examples**

Lock Spacing cset DEFAULT, which has a side effect of locking spacing and same net domains

```
axlCNSCsetLock('spacing "DEFAULT" t)
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **See Also**

[axICNSIsLockedDomain](#)

## **axlCNSDelete**

```
axlCNSDelete(  
    g_domain  
    t_name/o_dbidEcset  
)  
==> t/nil
```

### **Description**

Deletes a cset and its references to any objects such as nets, net classes, etc. Locked cssets must first be unlocked before you delete them. If it is a spacing or physical domain, you cannot delete the DEFAULT cset. You cannot delete electrical cssets in Allegro PCB Design L.

### **Arguments**

<i>g_domain</i>	Specifies the domain of cset. Valid values are: 'physical, 'spacing, 'sameNet, 'electrical
<i>t_name</i>	Name of cset.
<i>o_dbidEcset</i>	If an ECset, its <i>dbid</i> .

### **Value Returned**

<i>t</i>	Cset deleted.
<i>nil</i>	Cset not deleted because cset does not exist or the cset is locked, or cset is <i>t</i> .

### **Example**

Deletes electrical cset named UPREV\_DEFAULT.

```
axlCNSDelete('electrical "UPREV_DEFAULT")
```

### **See Also**

[axlCNSCreate](#)

## **axlCnsDeleteClassClassObjects**

```
axlCnsDeleteClassClassObjects(  
    ) => x_delCount
```

### **Description**

Delete all Class-Class entries.

### **Arguments**

None

### **Value Returned**

The count of the objects deleted.

### **See Also**

[axlCnsPurgeCsets](#)

## **axlCnsDeleteRegionClassClassObjects**

```
axlCnsDeleteRegionClassClassObjects(  
) => x_delCount
```

### **Description**

Deletes all Region-Class-Class entries.

### **Arguments**

None

### **Value Returned**

The count of the objects deleted.

### **See Also**

[axlCnsPurgeCsets](#)

## **axlCnsDeleteRegionClassObjects**

```
axlCnsDeleteRegionClassObjects(  
    ) => x_delCount
```

### **Description**

Delete all Region-Class entries.

### **Arguments**

None

### **Value Returned**

The count of the objects deleted.

### **See Also**

[axlCnsPurgeCsets](#)

## 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

<i>t_csetName</i>	Name of physical cset or nil for all csets.
<i>t_padstack</i>	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")
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **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  
  
axlCNSDesignModeGet(  
    s_name/t_name  
    'print  
) ==> t_name/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 `axlCNSDesignModeGet()` command.

The 'print mode offers the name shown in reports like show element.

This has [axlDebug](#) support.

**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 Allegro 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.
'print	Printable constraint name option.

## 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
<i>t_name</i>	The printable constraint name

## 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")
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

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/smode
```

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 1303.)

**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 Allegro 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/skillcmds
```

## axlCNSDesignValueCheck

```
axlCNSDesignValueCheck(  
    s_name/t_name  
    g_value  
)  
⇒ (t_string/nil, nil/tErrorMsg)/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</i> /nil)	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>tErrorMsg</i> )	Value incorrect. <i>tErrorMsg</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]  
)  
⇒ l1s_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>s_name</code>	Symbol name of value.
<code>t_name</code>	String name of value.
<code>g_returnNameString</code>	Returns constraint names as strings (default is symbol return.)
<code>g</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>l1s_constraintValues</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.

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **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

<i>t</i>	Design constraint value set.
<i>nil</i>	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
```

## **axlCNSDFAExport**

```
axlCNSDFAExport(t_fileName
    ) -> t_dfaFile
```

### **Description**

Saves current attachment file.

### **Value Returned**

<i>t</i>	if successful
nil	failed

### **See Also**

[axlCNSDFAImport](#)

### **Example**

Export DFA attachment file

```
ret = axlCNSDFAExport()
```

## **axlCNSDFAImport**

```
axlCNSDFAImport(  
    t_dfaFileName/nil  
) -> t/nil
```

### **Description**

This assigns or re-assigns a new DFA table to the design. Finds the file using DFACNSPATH and assumes a .dfa extension.

### **Arguments**

<i>t_name</i>	Name of a DFA file. If nil DFA table is removed from the design
---------------	---

### **Value Returned**

<i>t</i>	If successful
nil	Failed or if deleting attachment it does not exist

### **See Also**

[axlCNSDFAExport](#), [axlCNSDFAMode](#)

### **Examples**

Assigns a DFA table file called mfg.dfa

```
axlCNSDFAImport ("mfg")
```

## axlCNSDFAMode

```
axlCNSDFAMode(  
    s_option  
    [s_DRCMode]  
) -> t/nil/s_mode
```

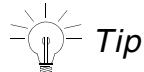
### Description

The function sets or queries the DFA mode. This accesses DFA options.

The *s\_option* settings are:

'table	Returns name of DFA table (for example, filename)
'mode	Returns the current DRC mode
'mode with an argument of 'on, 'off or 'batch	Sets the DFA DRC mode

If DRC mode is enabled DRC is set to out of date.



Use `axlCNSDFAMode ('table')` to test if a DFA table exists in design.

### Arguments

<i>s_option</i>	See above
<i>s_DRCMode</i>	Optional argument for mode (see above)

### Value Returned

t	if successful
nil	failed

For 'mode can return 'on, 'off or 'batch. For 'table returns DFA table name (this is DFA\_TABLE\_NAME attribute contained in the dfa file. It may not always match original filename).

## See Also

[axlCNSDFAImport](#)

## Examples

1. Query if DFA table exists with design

```
ret = axlCNSDFAMode ('table')
```

2. Get current DRC mode of DFA

```
ret = axlCNSDFAMode ('mode')
```

3. Set current DRC mode of DFA to on

```
ret = axlCNSDFAMode ('mode' 'on')
```

## **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 (Changed to upper case) String must pass allowed character set.
<i>t_copyName</i>	Optional name to copy from.

### **Value Returned**

<i>o_dbidEcset</i>	<i>dbid</i> of the new ECset
nil	Failed due to one of the following: the name is illegal, or the ECset already exists.

### **See Also**

[axlCNSDesignModeSet](#), [axlCNSCreate](#)

### **Example 1**

```
axlCNSEcsetCreate("MyEmptyEcset")
```

Creates a new empty ECset.

### **Example 2**

```
p = car(axlDBGetDesign() ->ecsets)
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

```
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 Allegro 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 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 electrical cset when you request it by the ECset name. Electrical Constraint Set (ECset) is a mechanism for grouping a set of electrical constraints and applying them to a set of nets.

### **Arguments**

*t\_name*                    ECset name.

### **Values Returned**

<i>o_dbidEcset</i>	<i>dbid</i> of the ECset requested.
<i>nil</i>	Function failed due to an illegal name.

### **See Also**

[axlCNSEcsetValueGet](#) and [axlCnsList](#)

### **Example**

```
axlCNSEcsetGet("foo")
```

Tests for the existence of an ECset named foo.

## axlCNSEcsetModeGet

```
axlCNSEcsetModeGet(  
    nil  
)  
⇒ ls_constraints  
  
axlCNSEcsetModeGet(  
    'all  
)  
⇒ lls_constraintNModes  
  
axlCNSEcsetModeGet(  
    s_name/t_name  
)  
⇒ s_mode/nil  
  
axlCNSEcsetModeGet(  
    s_name/t_name  
    'print  
) ==> t_name/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.

The 'print mode offers the name shown in reports, similar to the show element command.

This has [axlDebug](#) support.

**Note:** Not all checks are available in all levels of Allegro PCB Editor. To determine the set of checks supported, use the command: `axlCNSEcsetModeGet ()`. Constraint checks may change from release to release.

### Arguments

nil	Returns all checks in design type DRC.
'all	Returns all checks and current mode.
s_name	Symbol name of the check.
t_name	String name of the check.
'print	Printable constraint name option.

## Value Returned

<i>ls_names</i>	List of checks ( <i>s_name</i> ...).
<i>l1s_names</i>	List of checks and related modes (( <i>s_name s_mode</i> ) ...)
<i>s_mode</i>	Returns mode 'on, 'off, or 'batch
<i>t_name</i>	Printable constraint name.

## See Also

[axlCNSEcsetModeSet](#), [axlCNSEcsetValueGet](#)

### 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 Allegro 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 1303 for a set of interfaces you can use to mark changes in order to perform fewer DRC updates.



**Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant from release to release.**

## 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 Allegro 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.

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **Example 4**

```
axlCNSEcsetModeSet( '((Maximum_Crosstalk off)
    (Propagation_Delay on) (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 `axlCNSEcseValueGet (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**

<i>s_name</i>	Symbol name of constraint.
<i>t_name</i>	String name of constraint.
<i>g_value</i>	Value to verify.

### **Value Returned**

<i>t</i>	Syntax is correct.
<i>t_errorMsg</i>	Syntax is incorrect. The message indicates the reason.
<i>nil</i>	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 Allegro 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</i>	
<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>l1s_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)
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

```
= 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).

## **axlCNSGetDefaultMinLineWidth**

```
axlCNSGetDefaultMinLineWidth(
    t_subclassName
)
=> f_minLineWidthValue
```

### **Description**

Retrieves the minimum default line width value for the specific subclass.

### **Arguments**

*t\_subclassname* A subclass name of the ETCH or CONDUCTOR class.

### **Value Returned**

*f\_minSpacingValue* Minimum line width value (in design units) on the subclass.

### **Example**

```
axlCNSGetDefaultMinLineWidth("TOP")
=> 0.004
```

Gets the minimum line width value for layer TOP.

## **axlCNSGetPhysical**

```
axlCNSGetPhysical(
  t_cset
  t_layer
  s_constraint
  [g_string]
)
==> g_value/nil

axlCNSGetPhysical(
  t_cset
  t_layer
  nil
  [g_string]
)
==> ll_nameValue/nil

axlCNSGetPhysical(
  nil
  nil
  nil
)
==> ls_cnsTypes
```

### **Description**

In its first operational mode, obtains the value of a physical constraint given a cset and layer. In the second mode of operation, it obtains all physical constraint as name/value pairs for a cset on a layer. This, in turn, may be passed to `axlCNSSetPhysical`.

In the final mode, a list of all supported physical constraints may be obtained by passing three `nil` values to the interface:

```
axlCNSGetPhysical(nil nil nil)
```

### **Data types**

Unless otherwise specified, constraints are in current design units.

allow_etch	(boolean) <code>t/nil</code>
allow_ts	(symbol) <code>NOT_ALLOWED</code> , <code>ANYWHERE</code> , <code>PINS_ONLY</code> , <code>PIN_VIAS_ONLY</code>
allow_padconnect	(symbol) <code>ALL_ALLOWED</code> , <code>VIAS_PINS_ONLY</code> , <code>VIAS_VIAS_ONLY</code> , <code>NOT_ALLOWED</code>

## Allegro SKILL Reference

### Constraint Management Functions

---

vias	(string) colon separates the list of via names. Vias are not layer dependent, so are only returned for TOP. Order is important for etch editing working layer model. Use <a href="#">axlCnsGetViaList</a> to get the via list as a list of strings.
	When <code>width_max</code> , <code>dp_neck_gap</code> , <code>dp_primary_gap</code> , and <code>necklength_max</code> are set to 0, it indicates that this value is not used.

## Arguments

<code>t_cset</code>	Name of a physical cset. Can use "" for "DEFAULT".
<code>t_layer</code>	ETCH layer name (for example, "ETCH/TOP" or "TOP"). If <code>nil</code> , applies the change to all layers.
<code>s_constraint</code>	Name of constraint. If <code>nil</code> , returns a set of symbol/value pairs of all constraints.
<code>g_string</code>	By default, returns value in the native units of the constraint. If <code>g_string</code> is <code>t</code> , always returns data as a string.

## Value Returned

<code>g_value</code>	Value of constraint in design units, except for <code>same_net</code> , which is returned as a <code>t nil</code> .
<code>ll_nameValue</code>	Name values of pairs of physical constraint symbol and constraint value for all physical ( <code>s_constraint g_value</code> ). <code>'((necklength_min 10.0) (neckwidth_max 5.0) ...)</code>
<code>ls_cnsTypes</code>	List of supported physical constraint names.
<code>nil</code>	Returns <code>nil</code> on error (or <code>allow_etch</code> ).

## Example 1

```
axlCNSGetPhysical("") "TOP" 'width_min)
```

Gets the minimum line width in the default cset, TOP layer.

## Example 2

```
axlCNSGetPhysical("VOLTAGE" "BOTTOM" nil)
```

## Allegro SKILL Reference

### Constraint Management Functions

---

Gets all physical constraints for the DEFAULT, BOTTOM layer

#### **Example 3**

```
axlCNSGetPhysical("") "TOP" 'vias)
```

Gets the via list for default cset.

#### **Example 4**

```
axlCNSGetPhysical(nil nil nil)
```

Gets supported physical constraint symbols.

#### **Example 5**

```
cset = ""           ;; DEFAULT cset
foreach(subclass axlSubclassRoute()
layer = axlCNSGetPhysical(cset subclass nil)
printf("\nLAYER=%s\n\tconstraints=%L\n" subclass, layer)
)
```

Fetches all layers and constraints of physical cset DEFAULT.

#### **See Also**

[axlCNSSetPhysical](#), [axlCnsList](#), [axlSubclassRoute](#), and [axlCnsGetViaList](#)

## **axlCNSGetPinDelayEnabled**

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

### **Description**

Returns if pin delay is enabled.

### **Arguments**

None

### **Value Returned**

t:	Pin delay is enabled.
nil:	Pin delay is not enabled.

## **axlCNSGetPinDelayPVF**

```
axlCNSGetPinDelayPVF()  
=> t_pinDelayPVF
```

### **Description**

Returns the pin delay propagation velocity factor.

### **Arguments**

None

### **Value Returned**

*t\_pinDelayPVF*      If the pin delay propagation velocity factor is defined, it is returned as a string. If not defined, a blank string is returned.

## **axlCNSGetSameNet**

```
axlCNSGetSameNet (
  t_cset
  t_layer
  s_constraint
  [g_string]
)
==> g_value/nil

axlCNSGetSameNet (
  t_cset
  t_layer
  nil
  [g_string]
)
==> ll_nameValue/nil

axlCNSGetSameNet (
  nil
  nil
  nil
)
==> ls_cnsTypes
```

### **Description**

Obtains a same net spacing cset values. Documentation same as [axlCNSGetSpacing](#).

## Arguments

<i>t_cset</i>	Name of a same net spacing cset. Can use "" for "DEFAULT". If <i>nil</i> , return the list of all supported same net constraint symbols.
	<b>Note:</b> The value of the other arguments is ignored if <i>t_cset</i> is <i>nil</i> .
<i>t_layer</i>	ETCH layer name ( "ETCH/TOP" or "TOP"). <i>t_layer</i> cannot be <i>nil</i> or a blank or empty string.
<i>s_constraint</i>	Name of constraint. If <i>nil</i> returns a set of symbol/value pairs of all constraints for the layer specified by <i>t_layer</i> .
<i>g_string</i>	By default returns value in the native units of the constraint. If <i>g_string</i> is <i>t</i> , it will always return data as a string.

## Value Returned

<i>g_value</i>	Value of constraint in design units
<i>ll_nameValue</i>	Name value pairs of spacing constraint symbol and constraint value for all spacing. ( <i>s_constraint g_value</i> ). ' ((shape_shape 10.0) (line_line 5.0) ...)
<i>ls_cnsTypes</i>	List of supported same net spacing constraint names.
<i>nil</i>	Returns <i>nil</i> on error (or same_net).

## See Also

[axlCNSSetSameNet](#), [axlCnsList](#), [axlCNSGetSpacing](#)

## Examples

- Get shape to shape same net spacing in default cset, TOP layer  

```
axlCNSGetSameNet("") "TOP" 'shape_shape)
```
- Get all same net constraints for 25\_MIL\_SPACE, BOTTOM layer  

```
axlCNSGetSameNet("25_MIL_SPACE" "BOTTOM" nil)
```
- Get all same net constraints for default BOTTOM layer as strings  

```
axlCNSGetSameNet("") "BOTTOM" nil t)
```

## Allegro SKILL Reference

### Constraint Management Functions

---

- Get supported same net constraint symbols

```
axlCNSGetSameNet(nil nil nil)
```

- Fetch all layers and constraints of same net cset DEFAULT

```
cset = ""          ;; DEFAULT cset
foreach(subclass axlSubclassRoute()
layer = axlCNSGetSameNet(cset subclass nil)
printf("\nLAYER=%s\n\tconstraints=%L\n" subclass, layer)
)
```

## **axlCNSGetSameNetXtalkEnabled**

axlCNSGetSameNetXtalkEnabled() => *t/nil*

### **Description**

Returns if Same Net Xtalk is enabled.

### **Arguments**

None

### **Value Returned**

*t* : same net Xtalk is enabled.

*nil* : same net Xtalk is not enabled.

## **axICNSGetSpacing**

```
axlCNSGetSpacing(
    t_cset
    t_layer
    s_constraint
    [g_string]
)
==> g_value/nil

axlCNSGetSpacing(
    t_cset
    t_layer
    nil
    [g_string]
)
==> ll_nameValue/nil

axlCNSGetSpacing(
    nil
    nil
    nil
)
==> ls_cnsTypes
```

## Description

In its first operational mode, obtains the value of a spacing constraint given a cset and layer. All values are returned in design units, except for `same_net`, which is a boolean (`t/nil`). In a second mode of operation, it obtains all spacing constraints as name/value pairs for a cset on a layer. This, in turn, may be passed to `axlCNSSetSpacing`. For the final mode, a list of supported spacing constraints may be obtained by passing three `nil` values to this interface:

```
axlCNSGetSpacing(nil nil nil)
```

## Data types

- Unless otherwise specified, constraints are in current design units.

same\_net (boolean) *t / nil*

- `bbvia_gap` is not layer dependent. You must use the TOP layer name as the `t_layer` value to get or set this value.

## Arguments

<i>t_cset</i>	Name of a spacing cset. You can use "" for "DEFAULT".
<i>t_layer</i>	ETCH layer name (for example, "ETCH/TOP" or "TOP"). If <i>nil</i> , applies change to all layers.
<i>s_constraint</i>	Name of constraint. If <i>nil</i> , returns a set of symbol/value pairs of all constraints.
<i>g_string</i>	By default, returns value in the native units of the constraint. If <i>g_string</i> is <i>t</i> , always returns data as a string.

## Value Returned

<i>g_value</i>	Value of constraint in design units, except for <i>same_net</i> , which is returned as <i>t/nil</i> .
<i>ll_nameValue</i>	Name value pairs of spacing constraint symbol and constraint value for all spacing ( <i>s_constraint g_value</i> ).' <i>((shape_shape 10.0) (line_line 5.0) ...)</i>
<i>ls_cnsTypes</i>	List of supported spacing constraint names.
<i>nil</i>	Returns <i>nil</i> on error (or <i>same_net</i> ).

## Example 1

```
axlCNSGetSpacing("") "TOP" 'shape_shape)
```

Gets shape to shape spacing in default cset, TOP layer.

## Example 2

```
axlCNSGetSpacing("25_MIL_SPACE" "BOTTOM" nil)
```

Gets all spacing constraints for 25\_MIL\_SPACE, bottom layer.

## Example 3

```
axlCNSGetSpacing("") "BOTTOM" nil t)
```

Gets all spacing constraints for DEFAULT, bottom layer as strings.

## Example 4

```
axlCNSGetSpacing(nil nil nil)
```

Gets supported spacing constraint symbols.

### Example 5

```
cset = ""          ;; DEFAULT cset
foreach(subclass axlSubclassRoute()
layer = axlCNSGetSpacing(cset subclass nil)
printf("\nLAYER=%s\n\tconstraints=%L\n" subclass, layer)
)
```

Fetches all layers and constraints of spacing cset DEFAULT.

### See Also

[axlCNSSetSpacing](#), [axlCnsList](#), and [axlSubclassRoute](#)

## **axlCNSGetViaZEnabled**

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

### **Description**

Returns if Via Z is enabled.

### **Arguments**

None

### **Value Returned**

*t* : via Z is enabled

*nil* : via Z is not enabled

## **axlCNSGetViaZPVF**

```
axlCNSGetViaZPVF()  
=> t_viaZPVF
```

### **Description**

Returns the via Z propagation velocity factor

### **Argument**

None

### **Value Returned**

*t\_viaZPVF*: If the via Z propagation velocity factor is defined, it is returned as a string. If not defined, a blank string is returned.

## axlCNSPhysicalModeGet

```
axlCNSPhysicalModeGet(  
    nil  
) ==> ls_constraints  
  
axlCNSPhysicalModeGet(  
    'all  
) ==> lls_constraintNModes  
  
axlCNSPhysicalModeGet(  
    s_name/t_name  
) ==> s_mode/nil  
  
axlCNSPhysicalModeGet(  
    s_name/t_name  
    'print  
) ==> t_name/nil
```

### Description

This fetches the current physical DRC mode(s). Modes determine if a particular constraint is on or off. These modes apply to the entire board. To determine the set currently supported, physical modes do a `axlCNSPhysicalModeGet(nil)`. The physical mode set may be a subset of physical values since the implementation may associate certain values under a master mode. For example, `via_list` is not a constraint and the differential pair mode is under the `ecset` domain.

**Note:** Future releases may add or subtract constraint checks. The `axl` interface does guarantee the checks returned by this interface will remain constant from release to release.

## Arguments

<i>nil</i>	returns all modes that are in spacing domain
<i>all</i>	returns all checks and current mode
<i>s_name</i>	symbol name of check.
<i>t_name</i>	string name of check
<i>'print</i>	printable constraint name option

## 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, or 'off
<i>t_name</i>	the printable constraint name

## Examples

Get current list of physical constraints

```
axlCNSPhysicalModeGet(nil)
```

Get list of settings for all physical constraints

```
axlCNSPhysicalModeGet('all)
```

Get current mode of max line with

```
axlCNSPhysicalModeGet('width_max)
```

Get current setting of allow Ts using a string

```
axlCNSPhysicalModeGet("allow_ts")
```

## See Also

[axlCNSPhysicalModeSet](#), [axlCNSGetPhysical](#)

## **axlCNSIsCsetLocked**

```
axlCNSIsCsetLocked(  
    g_domain  
    t_csetName  
)  
==> t/nil
```

### **Description**

This returns if a cset is locked. See discussion in [axlCNSIsLockedDomain](#).

A locked cset has the following characteristics:

- Cannot be edited
- Has the effect of locking the entire domain

### **Argument**

<i>g_domain</i>	domain of cset; 'physical, 'spacing, 'sameNet, 'electrical
-----------------	---

### **Value Returned**

- *t* if cset is deleted
- *nil*, if cset is not deleted due to being locked or not a cset

### **Examples**

Command to check if the cset DEFAULT is locked

```
axlCNSIsCsetLocked('electrical "DEFAULT")
```

### **See Also**

[axlCNSDesignModeSet](#)

## **axlCNSIsLockedDomain**

```
axlCNSIsLockedDomain(  
    g_domain  
)  
==> t/nil
```

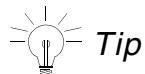
### **Description**

Used to check if the is constraint domain locked. A locked constraint domain has the following characteristics:

- csets cannot be edited, although new csets can be added
- any object (for example, net) level property overrides are ignored

### **Notes:**

- The spacing and sameNet domains are locked as a single domain.
- Locking is typically done via the techfile. In the techfile, you can lock individual csets. If one cset is locked, Allegro PCB Editor treats the entire domain as locked from the DRC perspective. When a domain is locked, any object level property constraint override is ignored.
- If a cset is locked it cannot be modified or deleted.



Use [axlCnsList\(nil\)](#) to get a list all domains.

### **Arguments**

<i>g_domain</i>	domain of cset; 'physical, 'spacing, 'sameNet, 'electrical
-----------------	---

### **Value Returned**

- t if a constraint domain is locked.
- nil - domain is not locked

## Examples

- Command to check if the Electrical domain locked

```
axlCNSIsLockedDomain('electrical)
```

- To find a list of locked domains

```
lockedDomains = setof( x axlCnsList(nil) axlCNSIsLockedDomain(x))
```

## See Also

[axlCNSCsetLock](#), [axlCNSIsCsetLocked](#), [axlCNSLockDomain](#), [axlCNSDesignModeSet](#),  
[axlCnsList](#)

## **axlCNSLockDomain**

```
axlCNSLockDomain(  
    g_domain  
    g_mode  
)  
==> t/nil
```

### **Description**

This command locks or unlocks a constraint domain.

See discussion in [axlCNSIsLockedDomain](#).

**Note:** Changing the lock on a domain can take a considerable amount of time since DRC status for that domain needs to be updated. In the spacing domain dynamic shapes also need to be updated. If doing other changes, you should consider cloaking (axIDBCloak the entire process. This API already uses cloaking.

### **Arguments**

<i>g_domain</i>	domain of cset; 'physical, 'spacing, 'sameNet, 'electrical
<i>g_mode</i>	may either be <code>t</code> (to lock) or <code>nil</code> to unlock

### **Value Returned**

Returns `t` if lock status is updated, and `nil` in case of an error.

### **Examples**

Lock Spacing and Same net spacing domains

```
axlCNSLockDomain('spacing t)
```

### **See Also**

[axlCNSIsLockedDomain](#)

## axlCNSOptions

```
axlCNSOptions (
    s_name
    [g_value]
)
==> g_currentValue/ls_names
```

### Description

Inquires and/or sets the value for constraint options. Typically, setting options will mark DRC out of date. It is calling code to update DRC.

Items currently supported:

Name	holeAlways
Value	t to nil
Set?	Yes
Description	Reports if hole check should always be done. Default is only to do the hole check if the pad is suppressed.
Equiv	<i>Analysis Modes – Spacing Options</i>
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

See above.

ls_names	If name is nil then returns a list of all controls
----------	--

### See Also

[axlCNSDesignModeSet](#), [axlDBControl](#)

### Examples

Check hole always

## **Allegro SKILL Reference**

### Constraint Management Functions

---

```
old = axlDBControl('holeAlways, nil)
```

## axlCNSPhysicalModeSet

```
axlCNSPhysicalModeSet(
    t_name/s_name
    t_mode/s_mode
)
==> t/nil

axlCNSPhysicalModeSet(
    'all
    t_mode/smode
)
==> t/nil

axlCNSPhysicalModeSet(
    l_constraintNModes
    t_mode/smode
)
==> t/nil

axlCNSPhysicalModeSet(
    ll_constraintNModes
)
==> t/nil
```

### Description

This sets the current DRC modes (on/off) for checks in the area of physical constraints. These modes are global. To determine the constraints modes currently supported do a axlCNSPhysicalModeGet(nil). We support several interfaces. All checks may be set ('all), individual checks, (t\_name), list of checks with a same mode' (s\_name ...) t\_mode/s\_mode' (t\_name ...) t\_mode/s\_mode and sets of checks via a list of: '((s\_name/t\_name s\_mode/t\_mode) ....)

The constraints names may be passed as a symbol or a string. For performance reasons, you should either do all your updates in a single call or wrap individual changes in the map API (see axlCNSMapUpdate).

**Note:** Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant 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 or 'off.
<i>t_mode</i> :	string mode setting "on or "off".
'all:	set all checks for given tier of Allegro.

## Value Returned

Returns t if succeeds or nil if failure.

## See Also

[axlCNSPhysicalModeGet](#), [axlCNSGetPhysical](#), [axlCNSMapUpdate](#)

## Examples

Turn all constraints off

```
axlCNSPhysicalModeSet('all 'off)
```

Turn on line width max

```
axlCNSPhysicalModeSet('width_max 'on)  
Turn two constraint to on  
axlCNSPhysicalModeSet(' (bbvia_stagger_max bbvia_stagger_min) 'on)
```

Set various constraints to different modes

```
axlCNSPhysicalModeSet( '((width_max off) (allow_etch 'on)) )
```

## **axlCNSSameNetModeGet**

```
axlCNSSameNetModeGet(  
    nil  
) ==> ls_constraints  
  
axlCNSSameNetModeGet(  
    'all  
) ==> lls_constraintNModes  
  
axlCNSSameNetModeGet(  
    s_name/t_name  
) ==> s_mode/nil  
  
axlCNSSameNetModeGet(  
    s_name/t_name  
    'print  
) ==> t_name/nil
```

### **Description**

Same as [axlCNSSpacingModeGet](#).

## Arguments

<i>nil</i>	returns all modes that are in same net spacing domain
' <i>all</i>	returns all checks and current mode
<i>s_name</i>	symbol name of check.
<i>t_name</i>	string name of check
' <i>print</i>	printable constraint name option

## 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</i> <i>s_mode</i> ) ...)
<i>s_mode</i>	mode 'on, or 'off
<i>t_name</i>	The printable constraint name

## Examples

Get current list of same net spacing constraints

```
axlCNSSameNetModeGet(nil)
```

Get list of settings for all same net spacing constraints

```
axlCNSSameNetModeGet('all')
```

Get current setting of line to line

```
axlCNSSameNetModeGet('line_line')
```

Get current setting of line to shape using a string

```
axlCNSSameNetModeGet("line_shape")
```

## See Also

[axlCNSSameNetModeSet](#), [axlCNSGetSameNet](#), [axlCNSSpacingModeGet](#)

## **axlCNSSameNetModeSet**

```
axlCNSSameNetModeSet (
    t_name/s_name
    t_mode/s_mode
)
==> t/nil

axlCNSSameNetModeSet (
    'all
    t_mode/smode
)
==> t/nil

axlCNSSameNetModeSet (
    l_constraintNModes
    t_mode/smode
)
==> t/nil

axlCNSSameNetModeSet (
    ll_constraintNModes
)
==> t/nil
```

### **Description**

Same as axlCNSSSpacingModeSet.

### **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 or 'off.
<i>t_mode</i> :	string mode setting "on or "off".
<i>'all</i> :	set all checks for given tier of Allegro.

### **Value Returned**

Returns *t* if succeeds or *nil* if failure.

## See Also

[axlCNSSameNetModeGet](#), [axlCNSGetSameNet](#), [axlCNSSpacingModeSet](#)

## Examples

Turn off all same net spacing constraints

```
axlCNSSameNetModeSet('all 'off)
```

Turn on line to line check

```
axlCNSSameNetModeSet('line_line 'on)
```

Turn two constraints to on

```
axlCNSSameNetModeSet('((line_shape thrupin_line) 'on)
```

Set several constraints to different modes

```
axlCNSSameNetModeSet( '((line_line off)
                           (thrupin_shape on)) )
```

## **axlCNSSetPhysical**

```
axlCNSSetPhysical(  
    t_cset/nil  
    t_layer/nil  
    s_constraint  
    g_value  
)  
==> t/nil  
  
axlCNSSetPhysical(  
    t_cset/nil  
    t_layer/nil  
    ll_constraintValues  
    nil  
)  
==> t/nil
```

### **Description**

Allows updating physical constraint values. By passing nil at the appropriate argument, values for all csets and all layers may be changed.

### **Data types**

See axlCNSGetPhysical for the data type of each constraint.

Allowed Design Units:

- A number (integer or floating point) where units is current design units. Must not exceed accuracy of the design.
- Unitless string where accuracy cannot exceed database accuracy.
- String with units, data converted to current design units.

Allowed Data Values:

- Boolean: Use `t/nil` or `"true"/"false"`.
- Symbol: Use the symbol name or its string.

For best performance, when calling multiple axlCNS interfaces to update constraint values, wrap them in the `axlCnsMap` interfaces as shown below:

```
axlCNSMapClear()  
axlCNSSetPhysical(nil nil 'width_min 5)  
axlCNSSetPhysical("") nil 'allow_padconnect 'VIAS_PINS_ONLY)
```

```
...  
axlCNSMapUpdate()
```

Single change calls do not require this.

For a list of physical constraints, see `axlCNSGetPhysical`. If adding/deleting individual vias, you may find it easier to use `axlCnsAddVia` and `axlCnsDeleteVia`.



***Same\_net behavior will change in 16.2. This does not change override values. For example, you can set width\_min value in all csets, but if the you applied it to a net or constraint area as an override, it will still be used for those items.***

## Arguments

<code>t_cset</code>	Cset name. You can use "" for the DEFAULT cset. Use <code>nil</code> to apply changes to all csets.
<code>t_layer</code>	ETCH layer name (for example, "ETCH/TOP" or "TOP"). If <code>nil</code> , applies changes to all layers.
<code>s_constraint</code>	Constraint symbol to change. Use <code>axlCNSGetPhysical(nil nil nil)</code> for list of permissible values.
<code>g_value</code>	Value to update. For data types, see Data Types above.
<code>ll_constraintValues</code>	Multiple values may be updated by passing a list of lists for the third argument.  <code>'((s_constraint g_value) ... )</code>

## Value Returned

<code>t</code>	Success.
<code>nil</code>	An error occurs when the ECset name does not exist; the layer does not exist; the constraint does not exist; the value for the constraint is illegal; or the cset is locked.

## Example 1

```
axlCNSSetPhysical(nil nil 'width_min 5)
```

## Allegro SKILL Reference

### Constraint Management Functions

---

Sets minimum line width on all constraints and layers

#### **Example 2**

```
axlCNSSetPhysical("") nil 'allow_etch t)
```

Sets allow\_etch on all layers in default cset.

#### **Example 3**

```
axlCNSSetPhysical("VOLTAGE" "top" 'allow_ts "NOT_ALLOWED")
```

Doesn't allow Ts on top layer of VOLTAGE cset.

#### **Example 4**

```
axlCNSSetPhysical("VOLTAGE" "top" 'allow_ts 'NOT_ALLOWED)
```

Uses the same value.

#### **See Also**

[axlCNSGetPhysical](#), [axlCNSMapClear](#), [axlCNSMapUpdate](#) [axlCnsAddVia](#), and [axlCnsDeleteVia](#)

## axlCNSSetSpacing

```
axlCNSSetSpacing(  
    t_cset/nil  
    t_layer/nil  
    s_constraint  
    g_value  
)  
==> t/nil  
  
axlCNSSetSpacing(  
    t_cset/nil  
    t_layer/nil  
    ll_constraintValues  
    nil  
)  
==> t/nil
```

### Description

Allows updating spacing constraint values. By passing *nil* at the appropriate argument, values for all csets and all layers may be changed.

### Data types

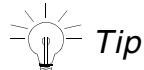
See [axlCNSGetSpacing](#) for the data type of each constraint.

Allowed Design Units:

- A number (integer or floating point) where units is current design units. Must not exceed accuracy of the design.
- Unitless string where accuracy cannot exceed database accuracy.
- String with units, data converted to current design units.

Allowed Data Values:

- Boolean: Use *t/nil* or "true"/"false".



For best performance, when calling multiple axlCNS interfaces to update constraint values, wrap them in the axlCnsMap interfaces as shown below:

```
axlCNSMapClear()  
axlCNSSetSpacing(nil nil 'line_shape 10.0)
```

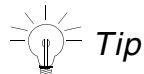
## Allegro SKILL Reference

### Constraint Management Functions

---

```
axlCNSSetSpacing("") nil 'line_line 5)  
...  
axlCNSMapUpdate()
```

Single change calls do not require this.



For a list of current spacing constraints, see [axlCNSGetSpacing](#).



***Same\_net constraint has been moved to the same net spacing domain.***



***An idiosyncrasy when values sent as strings requires the number of decimal points to be no more than the current database accuracy, or the change will be rejected.***



***This does NOT change override values. For example, you can change the line to line value in all csets but if you have applied a net or constraint area override, it will still be used for those items.***

## Arguments

<i>t_cset</i>	The cset name. You can use "" for DEFAULT cset. Use <i>nil</i> to apply the changes to all csets.
<i>t_layer</i>	The ETCH layer name (for example, "ETCH/TOP" or "TOP"). If <i>nil</i> , applies the changes to all layers.
<i>s_constraint</i>	Constraint symbol to change. Use <code>axlCNSGetPhysical(nil nil nil)</code> for a list of permissible values.
<i>g_value</i>	Value to update. For data types, see <a href="#">Data types</a> above.
<i>ll_constraintValues</i>	Multiple values may be updated by passing a list of lists for the third argument.  <code>'((s_constraint g_value) ...))</code>

## Value Returned

<i>t</i>	Success.
<i>nil</i>	Indicates an error. An error occurs when the ECset name does not exist; the layer does not exist; the constraint does not exist; the value for the constraint is illegal; or the cset is locked.

## Examples

- Set line to shape spacing in all csets, all layers  
`axlCNSSetSpacing(nil nil 'line_shape 5)`
- Set line to line spacing to 5 on DEFAULT cset, all layers  
`axlCNSSetSpacing("") nil 'line_line 5)`
- Value of DEFAULT cset  
`axlCNSSetSpacing("25_MIL_SPACE" "top" 'line_line 5)`
- Set a single spacing for all ids for all layers

```
cnsIds = axlCNSGetSpacing(nil nil nil)
values = nil
defaultSpace = 10.0
foreach( id cnsIds
    values = cons(list(id defaultSpace) values)
)
axlCNSSetSpacing("10_MIL_SPACE" nil values)
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **See Also**

[ax1CNSGetSpacing](#), [ax1CNSMapClear](#), and [ax1CNSMapUpdate](#)

## **axlCNSSetPinDelayEnabled**

axlCNSSetPinDelayEnabled(*g\_value*) => *t*

### **Description**

Enables or disables Pin Delay.

### **Argument**

*g\_value*: *t* or *nil* to indicate if Pin Delay is turned on or off.

### **Value Returned**

*t*

## **axlCNSSetPinDelayPVF**

```
axlCNSSetPinDelayPVF(g_value)
=> t/nil
```

### **Description**

Sets a value for pin delay propagation velocity.

### **Arguments**

*g\_value*: a string to define the new pin delay propagation velocity factor.  
A *nil* value indicates that the value is to be deleted.

### **Value Returned**

<i>t</i> :	no errors
<i>nil</i> :	error detected

## **axlCNSSetSameNet**

```
axlCNSSetSameNet(  
    t_cset/nil  
    t_layer/nil  
    s_constraint  
    g_value  
)  
==> t/nil  
  
axlCNSSetSameNet(  
    t_cset/nil  
    t_layer/nil  
    ll_constraintValues  
    nil  
)  
==> t/nil
```

### **Description**

Updates the same net spacing cset value. Documentation same as [axlCNSSetSpacing](#).

## Arguments

<i>t_cset</i>	Cset name, can use "" for DEFAULT cset. Use <code>nil</code> to apply change to all cset.
<i>t_layer</i>	ETCH layer name ( "ETCH/TOP" or "TOP"). If <code>nil</code> apply change to all layers.
<i>s_constraint</i>	Constraint symbol to change. Use <code>axlCNSGetSameNet(nil nil nil)</code> for list of permissible values.
<i>g_value</i>	Value to update. For data type, see above for "DATA TYPES".
<i>ll_constraintValues</i>	Multiple values may be updated by passing a list of lists for the third argument. <code>'((s_constraint g_value) ... )</code>

## Value Returned

<i>t</i>	if succeeds
<i>nil</i>	an error if any of the following occurred: <ul style="list-style-type: none"><li>■ cset name does not exist</li><li>■ layer does not exist</li><li>■ constraint does not exist</li><li>■ illegal value for constraint</li><li>■ cset is locked</li></ul>

## See Also

[axlCNSGetSameNet](#), [axlCNSSetSpacing](#)

## Examples

- Set line to same net spacing in all csets, all layers  
`axlCNSSetSameNet(nil nil 'line_shape 5)`
- Set line to line same net to 5 on DEFAULT cset, all layers  
`axlCNSSetSameNet("") nil 'line_line 5)`
- Value of DEFAULT cset  
`axlCNSSetSameNet("25_MIL_SPACE" "top" 'line_line 5)`

## **axlCNSSetSameNetXtalkEnabled**

```
axlCNSSetSameNetXtalkEnabled(g_value)
=> t
```

### **Description**

Enables or disables Same Net Xtalk.

### **Arguments**

*g\_value*: t or nil to indicate if same net Xnet is turned on or off.

### **Value Returned**

t

## **axlCNSSetViaZEnabled**

axlCNSSetViaZEnabledenabled(g\_value) => t

### **Description**

Enables or disables Via Z.

### **Arguments**

*g\_value*: t or nil to indicate if Via Z is turned on or off.

### **Value Returned**

t

## **axlCNSSetViaZPVF**

```
axlCNSSetViaZPVF(g_value)
=> t/nil
```

### **Description**

Sets a value for Via Z propagation velocity factor.

### **Arguments**

*g\_value*: a string to define the new via Z propagation velocity factor. A *nil* value indicates that the value is to be deleted.

### **Value Returned**

*t*: no errors

*nil*: error detected

## **axlCNSSpacingMax**

```
axlCNSSpacingMax(  
    [s_spacingType]  
    [t_layer]  
)  
=> f_maxSpacing
```

### **Description**

Returns maximum spacing in design. This can be worst case spacing of entire design or base upon the object filtering.

Maximum spacing is calculated from all spacings in the domains of spacing, same net, and assembly (APD+). The state of the check (on/off) is ignored (this is different from the minimum spacing API).

### **Arguments**

<i>s_spacingType</i>	Symbol representing the spacing constraint type. The supported values are: 'line, 'shape, 'via, 'pin
<i>t_subclassname</i>	A subclass name of the class ETCH or CONDUCTOR or nil for all layers

### **Value Returned**

<i>f_maxSpacing</i>	Maximum spacing on entire design or sub-filtered setting.
---------------------	---

### **Examples**

- Get maximum spacing on entire design  
`axlCNSSpacingMax(nil)`
- Get maximum spacing on lines (clines) layer TOP  
`axlCNSSpacingMax('line "TOP")`
- Get maximum spacing on lines (clines) all layers  
`axlCNSSpacingMax('line nil)`

## See Also

[axlCNSSpacingMin](#)

## **axlCNSSpacingMin**

```
axlCNSSpacingMin(  
    [s_spacingType]  
    [t_layer]  
)  
=> f_minSpacing
```

### Description

Returns minimum spacing value in the design. This can be the minimum spacing of entire design or based upon the object filtering.

Maximum spacing is calculated from spacings in the domains of spacing, same net, and assembly (APD+). The spacing check must be enabled (on) to be possibly included as a minimum.

### Arguments

<i>s_spacingType</i>	Symbol representing the spacing constraint type. The supported values are: 'line, 'shape, 'via, 'pin
<i>t_subclassname</i>	A subclass name of the class ETCH or CONDUCTOR or <i>nil</i> for all layers

### Value Returned

<i>f_minSpacing</i>	Minimum spacing on entire design or sub-filtered setting.
---------------------	---

### Examples

See Examples section of [axlCNSSpacingMax](#)

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **See Also**

[axlCNSSpacingMax](#)

## axlCNSSpacingModeGet

```
axlCNSSpacingModeGet(  
    nil  
) ==> ls_constraints  
  
axlCNSSpacingModeGet(  
    'all  
) ==> lls_constraintNModes  
  
axlCNSSpacingModeGet(  
    s_name/t_name  
) ==> s_mode/nil
```

### Description

This fetches the current spacing DRC mode(s). Modes determine if a particular constraint is on or off. These modes apply to the entire board. To determine the set currently supported spacing modes do a axlCNSSpacingModeGet(nil).

The spacing mode set may be a subset of spacing values since the implementation may associate certain values under a master mode.

**Note:** Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant from release to release.

## Arguments

<i>nil</i>	returns all modes that are in spacing domain
' <i>all</i>	returns all checks and current mode
<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, or 'off

## See Also

[axlCNSSpacingModeSet](#), [axlCNSGetSpacing](#)

## Examples

Get current list of design constraints

```
axlCNSSpacingModeGet(nil)
```

Get list of settings for all design constraints

```
axlCNSSpacingModeGet('all')
```

Get current setting of line to line

```
axlCNSSpacingModeGet('line_line')
```

Get current setting of line to shape using a string

```
axlCNSSpacingModeGet("line_shape")
```

## axlCNSSpacingModeSet

```
axlCNSSpacingModeSet(
    t_name/s_name
    t_mode/s_mode
)
==> t/nil

axlCNSSpacingModeSet(
    'all
    t_mode/smode
)
==> t/nil

axlCNSSpacingModeSet(
    l_constraintNModes
    t_mode/smode
)
==> t/nil

axlCNSSpacingModeSet(
    ll_constraintNModes
)
==> t/nil
```

### Description

This sets the current DRC modes (on/off) for checks in the area of spacing constraints. These modes are global. To determine the constraints modes currently supported do a axlCNSSpacingModeGet(nil). We support several interfaces. All checks may be set ('all), individual checks, (t\_name), list of checks with a same mode '(s\_name ...) t\_mode/s\_mode '(t\_name ...) t\_mode/s\_mode and sets of checks via a list of: '((s\_name/t\_name s\_mode/ t\_mode) ....) The constraints names may be passed as a symbol or a string. For performance reasons, you should either do all your updates in a single call or wrap individual changes in the map API (see [axlCNSMapUpdate](#)).

**Note:** Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant 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 or 'off.
<i>t_mode</i> :	string mode setting "on or "off"
'all':	set all checks for given tier of Allegro.

## Value Returned

Returns `t` if succeeds or `nil` if failure.

## See Also

[axlCNSSpacingModeGet](#), [axlCNSMapUpdate](#)

## Examples

Turn off all spacing constraints

```
axlCNSSpacingModeSet('all 'off)
```

Turn on line to line check

```
axlCNSSpacingModeSet('line_line 'on)
```

Turn two constraints to on

```
axlCNSSpacingModeSet(' (line_shape thrupin_line) 'on)
```

Set several constraints to different modes

```
axlCNSSpacingModeSet( ' ((line_line off)
                           (thrupin_shape on)) )
```

## **axlCnsPurgeAll()**

```
axlCnsPurgeAll(  
    ) -> x_purgeCount
```

### **Description**

Removes all unused constraint objects and constraint sets. Process all netclasses, regions, physical constraint sets and spacing constraint sets. Deletes all empty netclasses and regions.

### **Arguments**

None

### **Value Returned**

The count of the deleted items.

### **See Also**

[axlCnsPurgeCsets](#)

### **Examples**

```
axlCnsPurgeAll()
```

## **axlCnsPurgeCsets**

```
axlCnsPurgeCsets(  
    list l_type  
) -> x_purgeCount
```

### **Description**

Process all constraint sets of the specified domain and delete those without references.

This class of functions is design to help migrate designs to take advantage of the 16.0 constraint model. These functions do have to be used when migrating designs. Before using these functions you need to evaluate your constraint usage.

### **Arguments**

Domain of interest 'physical or 'spacing

### **Value Returned**

Count of the csets deleted.

### **Examples**

```
axlCnsPurgeCsets('physical)  
axlCnsPurgeCsets('spacing)
```

### **See Also**

[axlCnsPurgeObjects](#), [axlCnsPurgeAll\(\)](#), [axlCnsDeleteClassClassObjects](#),  
[axlCnsDeleteRegionClassClassObjects](#), [axlCnsDeleteRegionClassObjects](#)

## **axlCnsPurgeObjects**

```
axlCnsPurgeObjects(  
    list l_type  
) -> x_purgeCount
```

### **Description**

Process the database and delete all group\_type objects that have no members; a netclass with no nets, or a region with no shapes.

### **Arguments**

Domain of interest 'physical or 'spacing.

### **Value Returned**

Count of the objects deleted.

### **Examples**

```
axlCnsPurgeObjects('netclass)  
axlCnsPurgeObjects('region)
```

### **See Also**

[axlCnsPurgeCsets](#)

## axlViaZLength

```
axlViaZLength(  
    t_layer1  
    t_layer2  
    [g_inclusion]  
) -> f_length
```

### Description

Returns the via length from layer1 to layer2. The layer names can either be given as the ETCH subclass name (TOP) or given as the formal SKILL layer name ("ETCH/TOP").

This is the length used in the ViaZ option to several DRC checks.

By default, does not use the thickness of the layer1 or layer2 in its calculation.

### Arguments

<i>t_layer1</i>	start layer name
<i>t_layer2</i>	end layer name
<i>g_inclusion</i>	Optional inclusion in calculation. <ul style="list-style-type: none"><li>■ 'first': include thickness of <i>t_layer1</i></li><li>■ 'second': include thickness of <i>t_layer2</i></li><li>■ 'both': include both layers</li><li>■ 'nil': do not include either</li></ul>

### Value Returned

<i>f_length</i>	via length in design units
-----------------	----------------------------

### Examples

Get length from top to bottom (excluding top and bottom thickness)

```
axlViaZLength("TOP" "BOTTOM")
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **See Also**

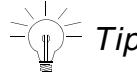
[axICNSGetViaZPVF](#)

## 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.



*Tip*  
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>	<code>dbid</code> 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](#)

## **Allegro SKILL Reference**

### Constraint Management Functions

---

#### **Examples:**

Net is part of an ECset (electrical constraint set) which has a MAX\_EXPOSED\_LENGTH constraint:

```
net = car(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 1303 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.

## Allegro SKILL Reference

### Constraint Management Functions

---

#### 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.

If a cset name is given, order of vias in list effects the via selection behavior of the etch editing's working layer model (see this documentation for more information).

Note that padstacks in via list may not currently be loaded in database or may not exist on disk (via that cannot be found is shown by a " \* " 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.

### **See Also**

[axlCnsAddVia](#), [axlCnsDeleteVia](#), and [axlCNSGetPhysical](#)

### **Examples**

Report vias in default physical constraint set

```
axlCnsGetViaList( "DEFAULT" )
```

Report vias in all physical constraint sets

## **Allegro SKILL Reference**

### Constraint Management Functions

---

`axlCnsGetViaList(nil )`

## **axl GetAllViaList**

```
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**

<code>[g_attrVias]</code>	Optional argument to add padstacks that are not included in constraint sets but are provided in some net VIA_LIST attributes.
---------------------------	---

### **Value Returned**

<code>lo_padstack_dbid</code>	List of padstack dbids.
<code>nil</code>	The design has empty via lists.

## **axlDRCUpdate**

```
axlDRCUpdate(  
    g_mode  
) -> x_cnt/nil
```

### **Description**

Performs a DRC check on entire design.

Has two return options controlled via `g_mode` option:

- `nil`: interactive (on) checks; similar to `drcupdate` command
- `t`: on and batch checks; similar to `dbdoctor drc` option

Will enable On-Line DRC if it is disabled. Obeys current DRC mode settings.



***Batch mode is being phased out.***

### **Arguments**

<code>g_mode</code>	<code>t</code> do all checks plus batch only checks, <code>nil</code> do only interactive checks
---------------------	--

### **Value Returned**

<code>x_cnt</code>	Returns number of errors
--------------------	--------------------------

### **See Also**

[axlDRCGetCount](#), [axlDBControl](#), [axlDRCWaive](#), [axlDBCheck](#)

### **Example**

Run a DRC check on a net named "GND"

```
db = axlDBFindByName('net "GND")  
cnt = axlDRCItem(nil p)
```

## **axlDRCWaive**

```
axlDRCWaive(  
    g_mode  
    o_DrcDbid/lo_DrcDbid  
    [t_comment]  
) ==> t/nil
```

### **Description**

Manages waive DRC state and access to the waive DRC functionality. It supports both waiving and restoring (unwaive) DRC markers. The interface supports both a single and a list of DRC *dbids*. If restoring a DRC marker, it will reappear but it may no longer reflect an actual DRC error. This may be due to:

- Change in the constraint expected value
- Change in the object(s) causing DRC
- Different DRC mode settings

The only way of determining if a DRC still should exist is to perform an `axlDRCItem` on the first item in the DRC's *dbid* violation attribute. The exception to this rule is external DRCs where the tool that created the DRC must be re-run. Note: Comment can also be added by adding the comment property to the DRC by:

```
axlDBAddProp(drcDbid '("COMMENT" "This drc is OK"))
```

## Arguments

<i>g_mode</i>	t: waive DRC. nil: unwaive DRC.
<i>o_DrcDbid</i>	A single DRC marker.
<i>lo_DrcDbid</i>	A list of DRC markers.
<i>t_comment</i>	Optional, add a comment to waived DRC. Only applies in waive mode.

## Values Returned

<i>t</i>	Success.
nil	Failed due to incorrect arguments.

## See Also

[axlDBControl](#), [axlDRCWaiveGetCount](#)

### Example 1 Waive 1st DRC in DRC list

```
p = axlDBGetDesign()->drcs  
axlDRCWaive(t car(p) "This DRC is OK")
```

### Example 2 Waive all DRCs in design

```
p = axlDBGetDesign()->drcs  
axlDBGetWaive(t p)
```

### Example 3 Restore all waived DRCs

```
p = axlDBGetDesign()->waived  
axlDBGetWaive(nil p)
```

## **axlDRCGetCount**

```
axlDRCGetCount(  
    )⇒ 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.

## axlDRCItem

```
axlDRCItem(  
    g_mode  
    o_dbid/lo_dbid  
)⇒ x_cnt/lo_drcDbid/nil
```

### Description

Performs a DRC check on the indicated item(s). The *dbid* may be any *dbid* type (except the design). If the same item appears multiple times in the list, then the same DRC error(s) are returned, and the count is the sum of errors created by each *dbid*. The *g\_mode* option controls two return options:

- |            |                             |
|------------|-----------------------------|
| <i>nil</i> | Returns DRC error count.    |
| <i>t</i>   | Returns list of DRC errors. |

This obeys current DRC mode settings, which includes the master DRC on/off switch.

Due to waive and duplicate DRC suppression processing, the list of DRCs returned using *g\_mode=t* may be less than the count returned by *g\_mode=nil*.



This is not an efficient way to run batch DRC or "what if" checks.

## Allegro SKILL Reference

### Constraint Management Functions

---

#### Arguments

<i>g_mode</i>	<i>nil</i> : Returns DRC error count. <i>t</i> : Returns list of DRC errors.
<i>o_dbid</i>	A single.

#### Value Returned

<i>x_cnt</i>	Returns number of errors associated with list of items.
<i>lo_drcDBid</i>	List of DRC <i>dbids</i> .
<i>nil</i>	No <i>dbids</i> (if <i>g_mode</i> = <i>t</i> ) or error in arguments.

#### See Also

[axlDRCGetCount](#), [axlDBControl](#), [axlDRCWaive](#), [axlDRCUpdate](#)

#### Examples

Run a DRC check on a net named "GND":

```
db = axlDBFindByName ('net "GND")  
cnt = axlDRCItem(nil p)
```

## **axIDRCWaiveGetCount**

```
axIDRCWaiveGetCount()  
    => x_count
```

### **Description**

Returns total number of waived DRCs in the design.

### **Arguments**

None.

### **Value Returned**

x\_count      Returns waived DRC count.

## **axILayerSet**

```
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

### **See Also**

[axlSetParam](#) and [axlLayerGet](#)

## Examples

## 1. Change color of top etch layer:

```
q = axlLayerGet("ETCH/TOP")
q->color = 7
q->pattern = 0           ; solid pattern
q->visibility = nil
axlLayerSet(q)
; if setting multiple layer colors/visisibility only call
; visible update after last change
axlVisibleUpdate(t)
```

2. To set all items to the same color on a class do

```
q = axlGetParam("paramLayerGroup:ETCH")
q->color = 7
axlSetParam(q)
axlVisibleUpdate(t)
```

## axlCnsList

```
axlCnsList(  
    s_csetDomain/nil  
)  
==> lt_csetNames/ls_csetsDomain
```

### Description

Returns the list of cset names of the domain specified. See `axlDBGetDesign()` ->`ecsets` for a list of electrical csets.

### See Also

[axlPurgePadstacks](#), [axlCnsDeleteVia](#), [axlCnsAddVia](#), and [axlCnsGetViaList](#)

### Arguments

<i>s_csetDomain</i>	Domains supported: spacing, physical, sameNet, and electrical.
<i>nil</i>	Lists all supported domains.

### Values Returned

<i>lt_csetNames</i>	Lists csets in specified domain.
<i>ls_csetDomains</i>	List of supported domains.

### See Also

[axlCNSCreate](#)

### Example 1

```
axlCnsList('spacing)
```

Returns all spacing cset names.

### Example 2

```
axlCnsList(nil)
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

Returns supported domains.

## **axlCNSMapClear**

```
axlCNSMapClear(  
)  
⇒ t
```

### **Description**

See [axlCNSMapUpdate](#).

### **Arguments**

none

### **Value Returned**

t	Always returns t.
---	-------------------

### **Examples**

See [axlCNSMapUpdate](#) on page 1303 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**

nil	There is no matching <code>axlCNSMapClear</code> .
<code>x_drcCount</code>	Number of DRCs caused by batch changes.

### **Examples**

- Turns off electrical max via check, turns all design checks on, and sets the island tolerance to 10. The Clear/Update calls batch up and optimize the DRC update required by these changes.

```
axlCNSMapClear()  
axlCNSEcsetModeSet('Maximum_Via_Count 'off)  
axlCNSDesignModeSet('all 'on)  
axlCNSDesignValueSet('Negative_Plane_Islands 10.0)  
axlCNSMapUpdate()
```

- Doing one change.

```
axlCNSMapClear()  
axlCNSEcsetModeSet('Maximum_Via_Count 'on)  
x1CNSMapUpdate()
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

For other real examples, see <cdsroot>/share pcb/examples/cmds files cns-design.il and acns\_design.form.

## 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`), Allegro 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. Allegro 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

<i>o_netDbid</i> /	dbid or name (string) of the net.
<i>t_netName</i>	
<i>t_cnsName</i>	Property name for the constraint.
<i>s_name</i>	Symbol name of DRC check (values returned by axlCNSEcsetModeGet(nil)). These names may not exactly match the property name.

## Value Returned

<i>t_cnsValue</i>	Value returned as a string exactall.
<i>nil</i>	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")
```

## DFM DRC Modes Functions

### axlCNSModeSet

```
axlCNSModeSet(
    t_domain/s_domain
    t_name/s_name
    t_mode/s_mode
)
⇒ t/nil

axlCNSModeSet(
    t_domain/s_domain
    'all
    t_mode/s_mode
)
⇒ t/nil

axlCNSModeSet(
    t_domain/s_domain
    l_constraintNModes
    t_mode/s_mode
)
⇒ t/nil

axlCNSModeSet(
    ll_domain
    ll_constraintNModes
)
⇒ t/nil
```

### Description

This function sets the current DRC modes for checks that fall into set of constraints belonging to a domain. These modes control the DRC for that check on the entire board.

To determine the set currently supported do a axlCNSModeGet().

This supports several interfaces.

All checks may be set to ('all), individual checks, (t\_name), list of checks with a same mode '(s\_name ...) t\_mode/s\_mode

'(t\_name ...) t\_mode/s\_mode

and sets of checks via a list of:

'((s\_name/t\_name s\_mode/t\_mode) ....)

For performance reasons, changing modes or values does not invoke DRC automatically. You need to run the DRC manually. A set of interfaces supports marking changes to perform a minimal set of DRC updates depending upon your changes (see axlCNSMapUpdate).



***Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant from release to release.***

## Arguments

<i>t_domain</i>	String name of the domain; may be "electrical", "design" "dff" "dfa", "dft" "spacing" "physical"
<i>s_domain</i>	Symbol name of the domain; may be 'electrical, 'design, 'dff, 'dfa, 'dft, 'spacing, 'physical
<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 "on", "off" or "batch"
'all	Set all checks for given tier of allegro in the specified domain.

## Value Returned

<i>t</i>	If successful
<i>nil</i>	If failed

## See Also

[axlCNSDesignModeGet](#), [axlCNSDesignValueGet](#), [axlCNSMapUpdate](#)

## Example

- Turn on pkg to pkg keepin check

```
axlCNSModeSet('design 'Package_to_Place_Keepin_Spacing 'on)
```

## Allegro SKILL Reference

### Constraint Management Functions

---

- Make all design constraints batch only

```
axlCNSModeSet('design 'all 'batch)
```

- Turn two constraint to off

```
axlCNSModeSet('design '(Negative_Plane_Islands Pad_Soldermask_Alignment)  
'off)
```

- Set various constraints to different modes

```
axlCNSModeSet('design '((Package_to_Place_Keepin_Spacing off)  
(Package_to_Place_Keepout_Spacing 'on)))
```

- Turn on all DFF constraints

```
axlCNSModeSet('dff 'all 'on)
```

- Turn off all DFT constraints

```
axlCNSModeSet('dft 'all 'off)
```

## axlCNSModeGet

```
axlCNSModeGet(
    t_domain/s_domain
    nil
)
⇒ ls_constraints

axlCNSModeGet(
    t_domain/s_domain
    'all
)
⇒ lls_constraintNModes

axlCNSModeGet(
    t_domain/s_domain
    'editable
)
⇒ t/nil

axlCNSModeGet(
    t_domain/s_domain
    s_name/t_name
)
⇒ s_mode/nil

axlCNSModeGet(
    t_domain/s_domain
    s_name/t_name
    'print
)
⇒ t_name/nil
```

### Description

This function fetches the current DRC modes for checks that fall into set of constraints under the specified domain. These constraints pertain to the entire board.

The 'print mode offers the name shown in reports like show element. This has axlDebug support.



***Future releases may add or subtract constraint checks. The axl interface does guarantee the checks returned by this interface will remain constant from release to release.***

## Arguments

<i>t_domain</i>	String name of the domain; may be "electrical", "design" "dff" "dfa", "dft" "spacing" "physical"
<i>s_domain</i>	Symbol name of the domain; may be 'electrical, 'design, 'dff, 'dfa, 'dft, 'spacing, 'physical
	nil: returns all checks in the DRC domain specified
	'all: returns all checks and current mode
	'editable: returns t if mode can be change, nil otherwise
	nil is returned when in Allegro studio which does not offer this option
<i>s_name</i>	Symbol name of check
<i>t_name</i>	String name of check
<i>'print</i>	Printable constraint name option

## 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
<i>t_name</i>	The printable constraint name

## See Also

[axlCNSDesignModeSet](#), [axlCNSDesignValueGet](#)

## Example

- Get current list of design constraints  

```
axlCNSModeGet( 'design nil)
```
- Get list of settings for all design constraints  

```
axlCNSModeGet( 'design 'all)
```
- Get current setting of pkg to pkg  

```
axlCNSModeGet('design 'Package_to_Package_Spacing)
```

## **Allegro SKILL Reference**

### Constraint Management Functions

---

- Get current setting of negative plane islands using a string

```
axlCNSModeGet("design" "Negative_Plane_Islands")
```

- Get the current settings of DFF constraints

```
axlCNSModeGet('dff 'all )
```

---

# **Command Control Functions**

---

The chapter describes the AXL-SKILL functions that register and unregister AXL-SKILL functions with Allegro PCB Editor and set various modes in the user interface.

## **AXL-SKILL Command Control Functions**

This section lists the command control functions.

## **axlCmdGetAppMode**

```
axlCmdGetAppMode ()  
-> t_appMode/nil
```

### **Description**

Retrieves name of the current application mode.

### **Arguments**

None

### **Value Returned**

nil	If no current application mode ("none" picked)
application mode	If one is set

### **Example**

```
axlCmdGetAppMode () ==> "generaledit"
```

## **axlCmdIsRegistered**

```
list axlCmdIsRegistered(  
    t_allegroCmd  
)  
⇒ 'user/'system/nil
```

### **Description**

Checks whether a command name is registered in the running application. Commands may be system commands or user-defined SKILL commands. If a command is already registered and you try to register it with `axlCmdRegister` using the same name, your command will replace the original. The original command will no longer be accessible in the system until your command has been unregistered again with `axlCmdUnregister`.

### **Arguments**

*t\_allegroCmd*      Name of command to be queried.

### **Value Returned**

'user	Command is registered as a user-defined SKILL command.
'system	Command is registered as a standard Allegro product command.
nil	Command is not registered.

### **Example**

```
(axlCmdIsRegistered "move")
```

### **See Also**

[axlCmdRegister](#), [axlCmdUnregister](#)

## axlCmdRegister

```
list axlCmdRegister(
    t_allegroCmd
    ts_callback
    ?cmdType t_cmdType
    ?doneCmd ts_doneCmd
    ?cancelCmd ts_cancelCmd
    ?undo t
)
⇒ t/nil
```

### Description

Registers a command named *t\_allegroCmd* with the Allegro PCB Editor shell system. If the command already exists, either because it is a base Allegro PCB Editor command or because it has been registered by this function at an earlier time, it will be hidden.

Additional arguments on the Allegro command line are passed unparsed to the AXL program.

You can call the `axlCmdRegister()` function at any time. Once a command is registered, you can type it at the Allegro PCB Editor command line and also incorporate it into menus and pop-ups.



Interactive mode should be used if you are changing the database. It will "done" an active interactive command before starting your command. General mode should be used to view the database or to provide some non-database capabilities. It is allowed to co-exist with other general and interactive commands. If changing the database using a form only command, for example, not requesting user picks from the canvas, then you should provide a dummy event handler to prevent your SKILL code from returning to Allegro.

See example in `<cdsroot>/share/pcb/examples/skill/cns-design.il`  
`function _AcDesignEvent().`

You can pass arguments from the Allegro command line to your registered SKILL function. See `<cdsroot>/share/pcb/examples/skill/examples/axlcore/arg.il`

Commands that support undo or redo must follow the programming concepts.

Programming checklist includes:

- utilize `axlDBTransaction` APIs around database changes
- do not save or open the databases

- do not call `axlRunBatchDBProgram` with an option to reload the database
- do not make use of `axlShell`
- if your program typically performs extensive database changes, do not register it as undoable due to memory consumption requirements

The following items do not follow undo or redo:

- files saved or updated on disk
- database attachment changes (see `axlCreateAttachment`)
- communication to external programs or user interfaces
- certain design state changes such as active layer
- if your program maintains state across invocations then redo may not work properly

An interactive command may be updated or written to support undo (see `cline2shape.il`) using two models:

- Undo all Allegro database changes made by the command from start to done. If you use database transactions in an existing SKILL command then to convert it to support undo all you need to do is to add the undo option to `axlCmdRegister`.
- To have it work like place manual, where each individual placement operation is an undo then you will need to revamp the transaction model of your code. You need to start and commit and a database transaction around each user change to the design. The `cline2shape.il` shows how the code is slightly different in transaction handling between the two modes.
  - Avoid doing a start transaction with the `undoMark` symbol and a commit if you do not perform database operations.
  - If you do this incorrectly, additional undo items are listed on the toolbar's undo icon. The code still performs properly but some undo operations will not do anything.



*Caution*

**You cannot access interactive Allegro PCB Editor commands using `axlShell` if you register your SKILL code as interactive, because nesting of interactive commands is not supported. This includes using `axlShell` to spawn Allegro PCB Editor scripts that contain interactive commands, which are those that display in the lower left side of the Allegro PCB Editor window (where the `Idle` string appears). For an interactive command, the status area shows the command name. Failing that, if you**

## **Allegro SKILL Reference**

### Command Control Functions

---

***notice odd behavior when calling axlShell, try changing your command type to General. As noted above, you should not call any APIs in the axlEnter or axlSelect families in a General command type.***

## Arguments

<i>t_allegroCmd</i>	Name of the command to register.
<i>ts_callback</i>	Name of SKILL callback routine called when the command is activated from the Allegro PCB Editor window.
<i>t_cmdType</i>	String denoting the type of this command: "interactive"  In which case all the auto-interactive rules for interactive commands are applied to this command. This is the default value.  "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 this type for pop-up commands.
<i>ts_doneCmd</i>	<i>Done</i> callback function that Allegro PCB Editor calls when the user types <i>done</i> or selects <i>Done</i> from the pop-up. Can be either a symbol or string. If <i>nil</i> , Allegro PCB Editor calls <i>axlFinishEnterFun</i> by default.  <b>Note:</b> The Done callback function is only registered for "interactive" command type. These arguments are ignored for any other command type.
<i>ts_cancelCmd</i>	Cancel callback function that Allegro PCB Editor calls when the user types <i>cancel</i> or selects <i>Cancel</i> from the pop-up. This argument can be either a symbol or a string. If it is <i>nil</i> , Allegro PCB Editor calls <i>axlCancelEnterFun</i> by default.  <b>Note:</b> The Cancel callback function is only registered for "interactive" command type. These arguments are ignored for any other command type.
<i>undo</i>	If <i>t</i> , command can be undone. It is only applicable for <i>t_cmdType</i> "interactive". Refer Tips section for programming requirements of undo commands.  Other options may be added at a future time.

## Value Returned

t	Command registered successfully.
nil	Command not registered.

## See Also

[axlCmdUnregister](#), [axlCmdList](#), [axlUIWHelpRegister](#), [axlDBTransactionStart](#), [axlCmdIsRegistered](#),

## Example

Registers the command `my swap gates` as calling the function `axlMySwapGates`.

```
axlCmdRegister("interboxcmd"
    'axlEnterBox
    "interactive"
    ?doneCmd 'axlBoxDone
    ?cancelCmd 'axlBoxCancel)
axlCmdRegister( "my swap gates" 'axlMySwapGates
?cmdType "interactive" ?doneCmd 'axlMySwapDone
?cancelCmd 'axlMySwapCancel)
```

For commands (`s_allegroCmd` value) that accept parameters as strings, the SKILL function converts the parameters to symbols. The following example is of a registered function that takes arguments:

```
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.

The following functions parse and process the `do it` arguments.

```
(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
    ) ; ;; 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)
```

## **Allegro SKILL Reference**

### Command Control Functions

---

```
(setq list (apply find_func nil))  
(print_list_props list p)  
(fprintf p "\n\n")  
 ) ; let  
) ;
```

See `<cdsroot>/share pcb/examples/skill/examples/axlcore/cline2shape.il` for the undo examples.

## **axlCmdUnregister**

```
list axlCmdUnregister(  
    t_allegroCmd  
)  
⇒ t/nil
```

### **Description**

Unregisters from the Allegro PCB Editor shell system, a previously registered command named `t_allegroCmd`. If the command already exists because it is a base Allegro PCB Editor command, the original command is available again. If the existing command was registered using `axlCmdRegister`, it will be lost.

### **Arguments**

`t_allegroCmd`      Name of command to be unregistered.

### **Value Returned**

`t`      Command unregistered successfully.

`nil`      Failed to unregister the specified command.

### **Example**

`axlCmdUnregister( "interboxcmd")` See Also

[axlCmdRegister](#), [axlCmdIsRegistered](#)

## **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

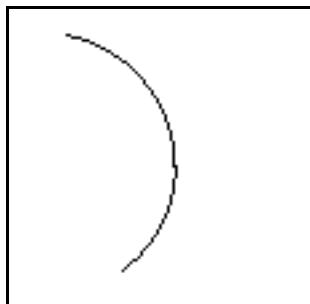
### **Value Returned**

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 Allegro PCB Editor window.



## **axlOKToProceed**

```
axlOKToProceed(  
)  
⇒ t
```

### **Description**

Checks whether Allegro 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 Allegro 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>	Allegro PCB Editor can allow AXL-SKILL database, selection, and interactive functions to execute.
<code>nil</code>	Allegro PCB Editor cannot allow AXL-SKILL database, selection, and interactive functions to execute.

### **Example**

```
(when axlOKToProceed  
  ; do AXL interactive command  
)
```

## **axlSetLineLock**

```
axlSetLineLock(  
    ?arcEnableg_arcEnable  
    ?lockAnglef_lockAngle  
    ?minRadiusf_minRadius  
    ?length45f_length45  
    ?fixed45g_fixed45  
    ?lengthRadiusf_lengthRadius  
    ?fixedRadiusg_fixedRadius  
    ?lockTangentg_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 Allegro PCB Editor Status form.

All parameters not explicitly set in a call to `axlSetLineLock` keep their current settings.

## Arguments

<i>g_arcEnable</i>	If t, sets Lock Mode to Arc. Otherwise Lock Mode is Line. Default is Line.
<i>f_lockAngle</i>	Sets Lock Direction. Allowed values are: 45 (degrees), 90 (degrees), or 0 (off, or no lock).
<i>f_minRadius</i>	Sets Minimum Radius, a value in user units.
<i>f_length45</i>	Sets the Fixed 45 Length value in user units.
<i>g_fixed45</i>	If t, sets the Fixed 45 Length mode. You cannot set this parameter unless <i>f_lockAngle</i> is 45, and <i>g_arcEnable</i> is nil.
<i>f_lengthRadius</i>	Sets Fixed Radius value in user units.
<i>g_fixedRadius</i>	If t, sets the Fixed Radius mode. You cannot set this parameter unless <i>f_lockAngle</i> is 45 or 90, and <i>g_arcEnable</i> is t.
<i>g_lockTangent</i>	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 Allegro 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: <code>done</code> or <code>cancel</code> .
<i>popupId</i>	List of the current pop-up values. A list of string pairs, as shown: <code>( ("Done" "axlFinishEnterFun") ("Cancel" "axlCancelEnterFun") )</code>
<i>ministatForm</i>	Always <code>nil</code> . (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 Allegro PCB Editor.
<i>nil</i>	Failed to get user data structure from Allegro 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 Allegro PCB Editor. If *r\_popup* is nil, unsets the currently active pop-up.

If this call is preceded by a call to axlUICmdPopupSet with a non-nil *r\_popup*, the contents of this popup are used to control graying of the popup items defined in the call to axlUICmdPopupSet. Otherwise *r\_popup* is used to replace the popup entries. In both cases, the popup callbacks will be replaced.

**Note:** The popup is invoked by pressing mouse popup button, this is normally the right mouse button.

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.

## 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**

<i>t</i>	Built form subclass pop-up.
<i>nil</i>	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 L and Allegro Package SI XL, 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 Allegro 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 owner drawn 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:** The *prettyprint* option upper/lower cases the popup name the user sees.

## Arguments

<i>r_form</i>	Standard form handle (see <a href="#">axlFormCreate</a> on page 819)
<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 = axlLayerGet("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.

---

## Polygon Operation Functions

---

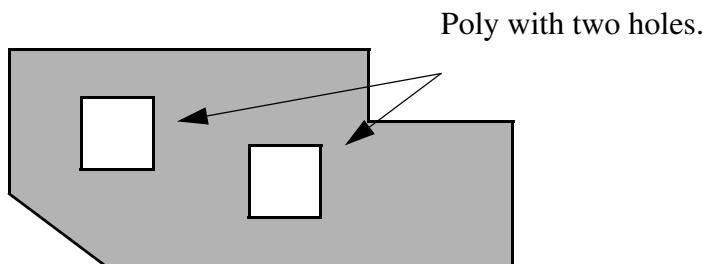
This chapter describes the AXL/SKILL Polygon Operation functions and includes the following sections:

- 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 21-1 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 Allegro PCB Editor unfilled shape.

These functions, which perform geometric operations on polys, are called logical operations and do the following:

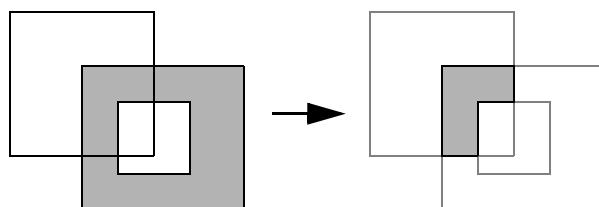
## Allegro SKILL Reference

### Polygon Operation Functions

- 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:

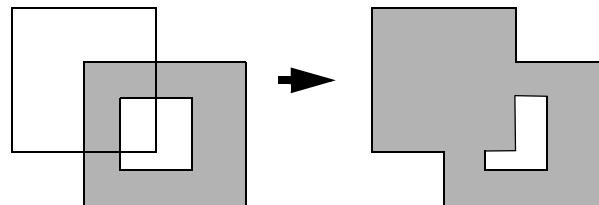
- Create logical operation objects (*lo\_polygon*) from these Allegro PCB Editor database objects:
  - pins
  - lines
  - clines
  - vias
  - shapes
  - rectangles
  - frectangles (filled rectangles)
  - voids
- Create *o\_polygon* from holes in a poly (*o\_polygon*)
- Create an Allegro PCB Editor database 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.



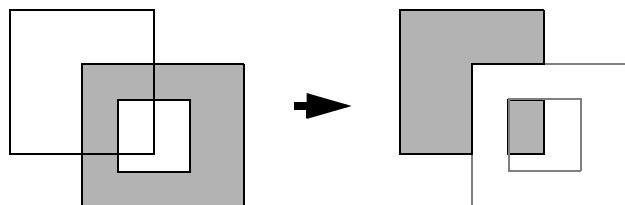
## Allegro SKILL Reference

### Polygon Operation Functions

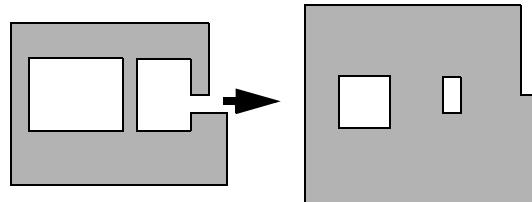
- ❑ Logical OR - The union of two polys.



- ❑ 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

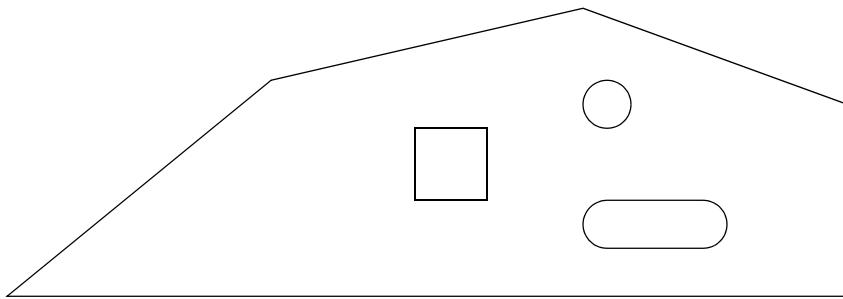
The following are the attributes of the *o\_polygon* type:

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>

## Allegro SKILL Reference

### Polygon Operation Functions

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/r_path
    ?endCapTypes_endCapType
    ?layer/_layer
    ?padType s_padType
    ?holes      t/nil
    ?line2poly  t/nil)
    ?xhatch t/nil)
    ?window nil/bBox)
⇒ lo_polygon
```

### Description

Creates a list of *o\_polygon* objects from the *dbid* or an *r\_path*. Use the *lo\_polygon* list to get the polygon attributes or to perform logical operations on these polygons. In the case of *r\_path* option, we expect a path that reflects a closed shape with no intersections. It is important that the first and last point be the same. The width option of *r\_path* is ignored as well as the '?' arguments to *axlPolyFromDB*.

### Polygon Attributes

area	(float) Area of polygon in design units. If a hole this is negative. If a polygon is not a hole then the area is the sum of the base poly area minus any of its holes.
bBox	(bBox) bounding box of polygon
holes	(list of o_polys/nil) list of any holes in polygon
isHole	(t/nil) is this a hole (void) or a shape
objType	(t_string) "polygon"

## Allegro SKILL Reference

### Polygon Operation Functions

---

vertices (list of coordinates). This always describes a closed shape.

Format for each coordinate is:

(xy f\_radius)

Where

xy - vertex point in design units

f\_radius - 0 if previous point and this point forms a segment else points form a arc with radius. The sign of the radius indicates for positive the arc is to the left of the y-axis and a negative indicates arc is to the right.

If arcs are present a polygon typically may contain more segments than the underlying shape dbid. This is due to the polygon arcs cannot cross a quadrant so are broken along quadrant boundaries.

#### NOTES

- A polygon is NOT a dbid.
- Comparing two polys using SKILL functions:
  - equal function: geometrically compares that two polygons are the same thus slightly slower than the eq function.
  - eq function compares that the polygon ids are the same.

This is different from the dbid comparison where both the `equal` and `eq` return the same results.

What this means from a programming standpoint, is that if you have two identical shapes in Allegro PCB Editor, the '`equal`' comparison on the shape dbids returns that they are NOT equal but converting these shapes to polys via `axlPolyFromDB` and doing a '`equal`' of the resulting polygons will return '`t`' while the '`eq`' comparison will return '`nil`'.

## Arguments

<i>o_dbid</i>	<p><i>axl dbid</i> for one of the following:</p> <p>path (line, cline or bondwire), shape, rect, frect, pin, via, void, bond finger, segment (arc or line), figure, text or DRC from which to construct the polygon.</p> <p><b>Note:</b> Arc and line are segments reported by <code>show element</code>.</p>
<i>r_path</i>	<p>Path construct from the <code>axlPath</code> API family. This is not an Allegro PCB Editor database object and is a much more efficient method for creating an Allegro PCB Editor shape, than converting it to a Poly. Note <code>axlDBCreateOpenShape</code> also supports an <i>r_path</i>. For more details, see <code>line2poly</code>. <i>r_path</i> must describe a closed non-intersecting shape unless the <code>line2poly</code> is <code>t</code> (see below).</p>
<i>s_endCapType</i>	<p>Keyword string specifying the end cap type to use for the polygon, one of '<code>SQUARE</code>', '<code>OCTAGON</code>', or '<code>ROUND</code>'. Used in case of line or cline only, otherwise ignored. Default is '<code>SQUARE</code>'.</p>
<i>t_layer</i>	<p>String specifying the layer of the pad to retrieve, for example, "<code>ETCH/TOP</code>". Used in the case of pin or via only, otherwise ignored. Default is "<code>ETCH/TOP</code>".</p>
<i>s_padType</i>	<p>Keyword string specifying the type of the pad to be retrieved, one of '<code>REGULAR</code>', '<code>ANTI</code>', or '<code>THERMAL</code>'. Used for pins, vias, or if <i>r_path</i> is based with the <code>line2poly</code> option, otherwise ignored. Default is '<code>REGULAR</code>'.</p>
<i>holes</i>	<p>Default value is <code>t</code>. If the value is set to <code>t</code>, for shapes with voids returns any voids as holes. If the value is set to <code>nil</code>, does not return the holes.</p>
<code>line2poly</code>	<p>Applicable only if first argument is an <i>r_path</i>. By default, an <i>r_path</i> describes a closed path. When this option is <code>t</code>, the <i>r_path</i> itself is converted to a polygon in a manner similar to <code>line dbids</code>. It is strongly recommended that the <i>r_path</i> has width otherwise the artwork undefined line width is used. Typically one polygon is returned for each segment in the <i>r_path</i>. If the value is set to <code>nil</code>, the <i>r_path</i> which describes a closed path is converted to that closed polygon. If the <i>r_path</i> does not describe a closed path, then this argument makes no difference. The default is <code>t</code>.</p>

## Allegro SKILL Reference

### Polygon Operation Functions

---

xhatch	If <i>t</i> and the dbid is a cross-hatch shape returns a polygon of the cross-hatching. By default, cross-hatch shapes are treated as solid shapes with respect to polygon generation. For complex shapes (>500 edges; this includes both outline and void boundaries) generating a poly of the cross-hatching can take a considerable amount of time and memory. If this option is <i>t</i> , the holes option for cross-hatch shapes is assumed to be <i>t</i> .
window	For converting shape objects, this option provides a bounding box to restrict the extents of the returned shape. Only the metal of the shape, which is inside this window is returned. For large complex shapes with many voids, this can be much more efficient than requesting the full poly and then using an axIPolyOperation call to cut it down to the area of interest. Default value is nil.

### Value Returned

<i>lo_polygon</i>	Object representing the resulting geometry.
nil	Cannot get polygons from the object.

### See Also

Other APIs that support or generate polygons:

- [axIPolyOperation](#) – performs various logical operations on 2 lists of polygon
- [axIPolyExpand](#) – expands or contracts polygon
- [axIIsPolyType](#) – is object a polygon object
- [axIPolyErrorGet](#) – returns last error from axIPolyOperation
- [axIPolyFromDB](#) – converts an allegro dbid to a polygon
- [axIPolyFromDrillHole](#) – return a poly for a drill hole
- [axIPolyMemUse](#) – debug function to return memory use of polygon sub-system
- [axIPolyOffset](#) – moves a polygon
- [axIPolyFromHole](#) – converts a hole polygon to a positive polygon
- [axIDBCreateShape](#) – creates a shape

See documentation for individual use.

## Examples

- Create a polygon from a via

```
polyList = axlPolyFromDB(via_dbid, ?layer "ETCH/BOTTOM" ?padType 'ANTI)
```

- Create a rectangle polygon (one corner at 0,0 with a width 1000 and height of 500) using *r\_path* method

```
; note first and last points are the same  
myPath = axlPathStart( list(0:0 1000:0 1000:500 0:500 0:0) 0)  
pathPoly = axlPolyFromDB(myPath )  
poly = car(pathPoly)  
poly->??
```

## **axlPolyFromHole**

```
axlPolyFormHole(  
    o_polygon  
)  
⇒ lo_polygon
```

### **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.

The *o\_polygon* must have isHole attribute set to t (that is the argument must be a hole).

### **Value Returned**

*lo\_polygon*                    Returns a list of *o\_polygons*, which represent the resulting geometry after creating polygon from the hole argument. In case of an error, it returns nil.

Creates an *lo\_polygon* object from a hole.

The *lo\_polygon* object can be manipulated with the following operations:

<i>axlPolyOperation()</i>	Performs various logical operations on 2 lists of polygons
<i>axlPolyExpand()</i>	Expands or contracts the size of polygon

### **Examples**

```
poly = axlPolyFromDB(shape_dbid)  
hole = car(poly->holes)  
polyList = axlPolyFromHole (hole)
```

## **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, for example, let or prog statements) then you need to insure all of global data is nil-ed at the end of your program. The following example 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 Polygons still in use.
- 2 - Number of Allegro Polys in use. This is always >= 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.

## **Allegro SKILL Reference**

### Polygon Operation Functions

---

```
gc() ; requires Skill development licenses  
axlPolyMemUse()  
;; 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**

*o\_polygon*                   *o\_polygon* on which the operation is to be done.

*lo\_polygon*                 Optionally pass a list of polys.

*l\_xy*                         Coordinates in user units for offset.

*g\_copy*                       Optional, if t does the offset on a copy.

### **Value Returned**

*lo\_polygon/*

*o\_polygon*                   In place offset (*g\_copy* nil) or offseted copy of polygon (*g\_copy* is t). If passed a list of polys returns a list otherwise return a poly.

### **See Also**

[axlPolyFromDB](#), [axlPolyRotate](#)

### **Example**

<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 polygons. Does not allow hole polygons as input. When holes are passed as input, the following warning is displayed:

Invalid polygon id argument -<argument>



***This function is provided "as-is". Result, in certain cases, may fail or deliver incorrect results. No commitment can be made to address issues uncovered when using this API.***



***Underlying polygon operation function fails and returns nil in rare dense geometrical situations.***



***This API may consume a large amount of memory and take a considerable of amount of time to return a result. This is normally only noticeable when the number of polygons provided exceed 10000. The number polygons can be calculated by taking the length of the polygons provided to args 1 and 2 PLUS adding all of the polys holes (poly->holes) in the polygons.***



***Algorithm has lines take have a width 1 or 0 database units. This means if you have a design with 2 units of accuracy, 1 database unit is .01.***

## Arguments

<i>o_polygon1</i> / <i>lo_polygon1</i>	<i>o_polygon</i> or list of <i>o_polygons</i> on which the operation is to be done.
<i>o_polygon2</i> / <i>lo_polygon2</i>	<i>o_polygon</i> or the list of <i>o_polygons</i> on which the operation is to be done.
<i>s_operation</i>	String specifying the type of logical operation, one of 'AND, 'OR, or 'ANDNOT.

## Value Returned

<i>lo_polygon</i>	List of <i>o_polygons</i> which represent the resulting geometry from performing the operation on the arguments.
<i>nil</i>	Error due to incorrect arguments.

For example:

(*o\_polygon\_out1* *o\_polygon\_out2* ...) is returned if the result after performing the operation is a list of polygons.

*nil* is returned if the result after performing the operation is a *nil* polygon. 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 logical operation function fails and returns nil in rare dense geometrical situations. 'ALL\_ARC mode may have round-off issues when shape has very small arc segments.***

Trimming options are (s\_expandType):

'NONE	no corner modifications
'ACU_ARC	Trim inside acute (less than 90 degrees) line/line corners with arcs and always chamfer spikes. No obtuse or right angle trimming is done.
'ACU_BLUNT	Trim acute inside corners and spikes with line segments.
'ALL_ARC	Trim inside and outside line/line, line/arc and arc/arc corners with respect to these angle rules: <ul style="list-style-type: none"><li>■ All acute angles are trimmed.</li><li>■ Most obtuse angles (more than 135 degrees) are trimmed.</li><li>■ 90 degree corners are trimmed.</li></ul> Finally always chamfer spikes.

**Note:** Poly expansion with 0 and no trim is returns the input poly NOT a list  
`axlPolyExpand(poly 0.0 'NONE) -> o_polygon.`

## 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, (see above)

## Value Returned

*lo\_polygon* List of *o\_polygons* which represent the resulting geometry after performing the expansion on the polys passed as arguments.

*nil* Failed to expand polys due to incorrect arguments.

To be more specific:

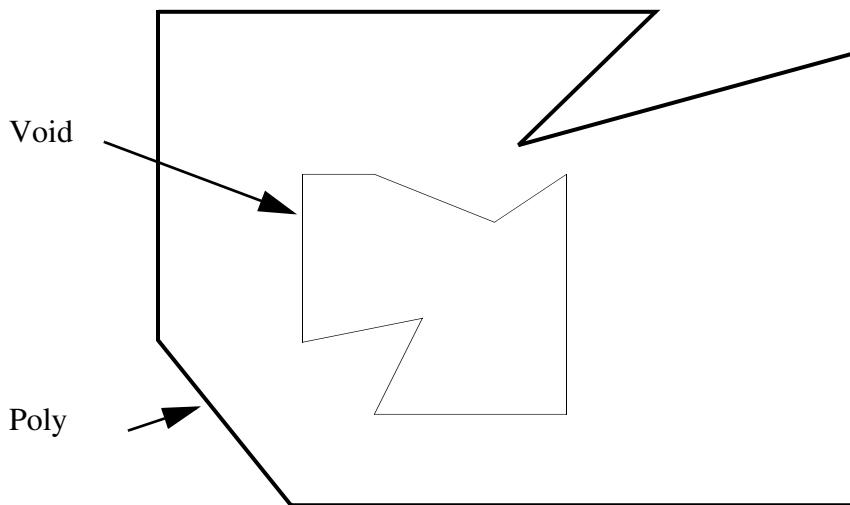
- (*o\_polygon\_out1 o\_polygon\_out2 ...*) is returned if the result after performing the operation is a list of polys.
- *nil* is returned if the result after performing the operation is a *nil* poly, for example, consider contracting a 20x30 rectangle by 40 units.
- *nil* is returned if the operation 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 21-2 Original Poly**



### **ALL\_ARC**

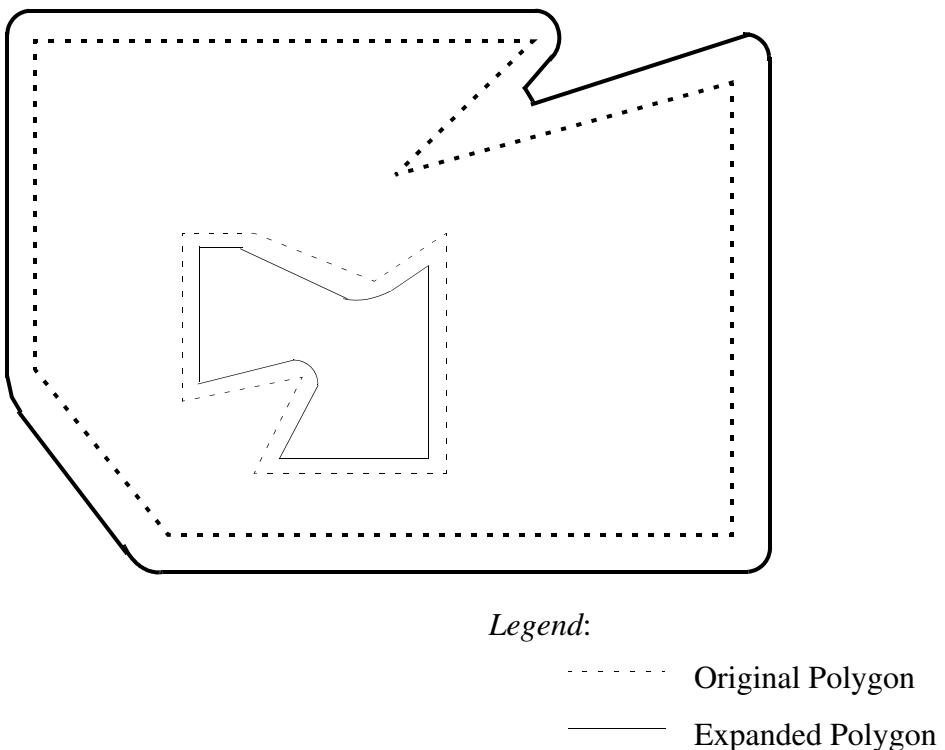
During expansion of the poly boundary, an arc is inserted for the edges in the offset shape that satisfy the following criteria:

- Edges form an *outside* (or convex) point of the poly boundary.  
The reverse is true for the voids.

## Allegro SKILL Reference

### Polygon Operation Functions

**Figure 21-3 Expanded Using ALL\_ARC**

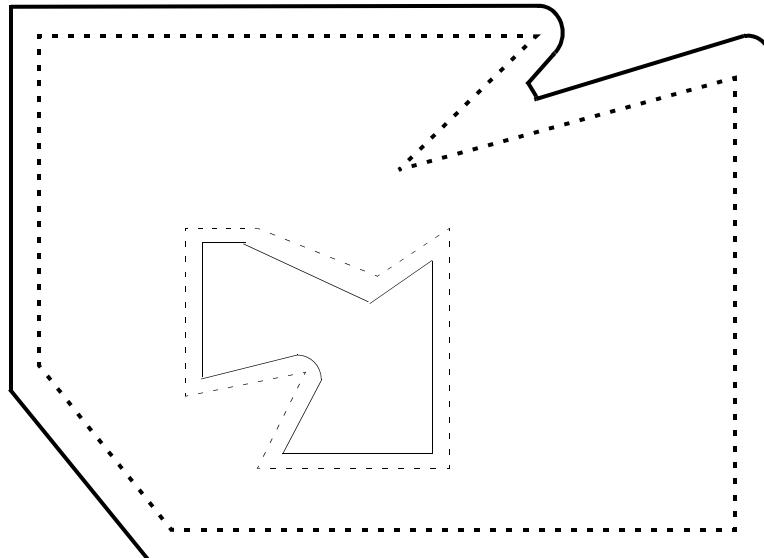


## **ACU\_ARC**

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 21-4 Expanded Using ACU\_ARC**



*Legend:*

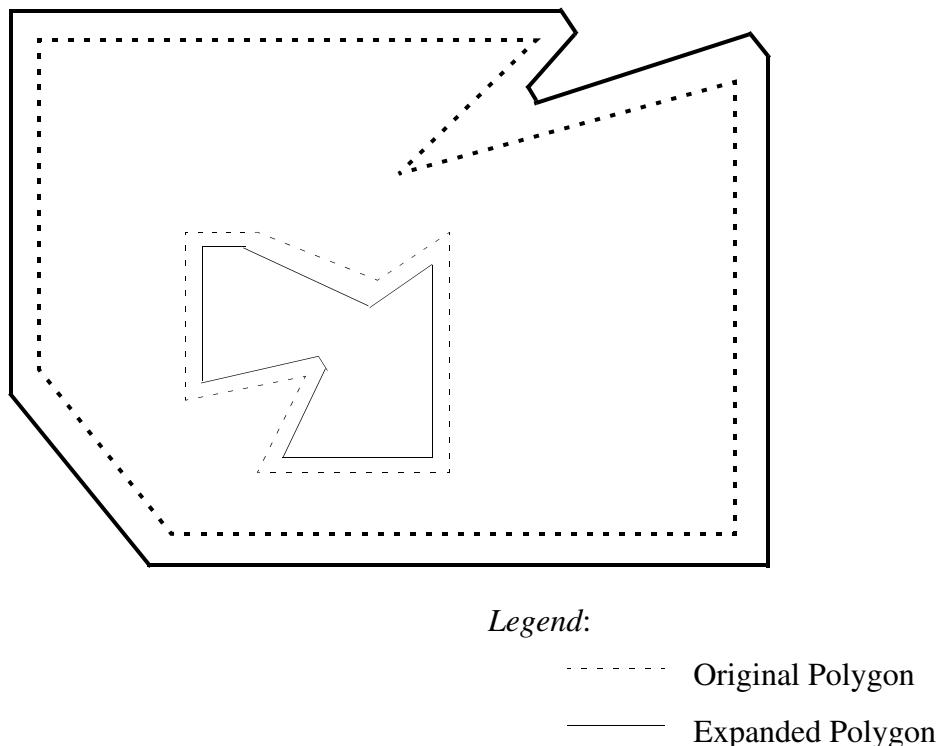
- Original Polygon
- Expanded Polygon

## **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 21-5 Expanded Using ACU\_BLUNT**



## **axlIsPolyType**

```
axlIsPolyType (
    g_polygon
)
⇒ t/nil
```

### **Description**

Tests if argument *g\_polygon* is a polygon user type.

### **Arguments**

<i>g_polygon</i>	Object to test, any SKILL variable
------------------	------------------------------------

### **Value Returned**

t	<i>g_polygon</i> is a polygon user type.
nil	<i>g_polygon</i> is not a polygon user type.

### **See Also**

[axlPolyFromDB](#)

### **Example**

```
poly = axlPolyFromDB(cline_dbid)
axlIsPolyType(poly) returns t.
axlIsPolyType(cline_dbid) returns nil.
```

## **axlPolyFromDrillHole**

```
axlPolyFromDrillHole (
  o_dbid
)
⇒ lo_polygon
```

### **Description**

Creates a polygon from a drill hole. Requires a via on pin *dbid*.

Function is restricted to do the following:

- Supports finished drill (future versions may add backdrill support)
- Multidrill returns a poly of the extent (future versions may add option to return a poly of each drill).

See [axlPolyFromDB](#) for polygon attributes.

### **Arguments**

*o\_dbid*                            *dbid* for pin or via

**Note:**

### **Value Returned**

*lo\_polygon*                            Returns the *lo\_polygon* object representing the resulting geometry.

*nil*                                    If object is not a pin or via or if lacks a drill hole

### **See Also**

[axlPolyFromDB](#)

### **Example**

Returns a drill polygon using the *ashOne* sample code that allows you to select a pin or via:

```
p = ashOne('("PINS" "VIAS")')
poly = car(axlPolyFromDrillHole(p))
```

## axlPolyFromHole

```
axlPolyFromDB (
    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

<i>o_polygon</i>	<i>o_polygon</i> on which the operation is to be done. The <i>o_polygon</i> must have <i>isHole</i> attribute set to <code>t</code> (that is, the argument must be a hole).
------------------	---

### Value Returned

<code>lo_polygon</code>	List of <i>o_polygons</i> which represent the resulting geometry after creating poly from the hole argument.
<code>nil</code>	Error due to incorrect argument.

### See Also

Creates an `lo_polygon` object from a hole. The `lo_polygon` object can be manipulated with the following operations:

- `axlPolyOperation()`: Performs various logical operations on 2 lists of polygon UDTs
- `axlPolyExpand()`: Expands or contracts

### Example

```
poly = axlPolyFromDB(shape_dbid)
hole = car(poly->holes)
polyList = axlPolyFromHole(hole)
```

## **axIPolyErrorGet**

```
axIPolyErrorGet (
)
⇒ t_error/nil
```

### **Description**

Retrieves the error from the logop core. See the following list of error strings returned by the logical operation core:

<b>Error type</b>	<b>String returned</b>
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

<i>t_error</i>	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())
```

## **axlPolyRotate**

```
axlPolyRotate(  
    o_polygon / lo_polygon  
    n_angle  
    l_origin  
    [b_copy]  
)  
⇒ o_polygon
```

### **Description**

Rotates a polygon or list of polygons. This function rotates the entire polygon by the provided angle around the given point. Optionally, if *g\_copy* is t it copies the polygon. Default is to rotate the provided polygon directly.

The rotated polygon must be entirely within the extents of the drawing.

### **Arguments**

<i>o_polygon</i> / <i>lo_polygon</i>	A polygon or list of polygons on which the operation is to be performed
<i>n_angle</i>	Rotation angle to be applied in degrees
<i>l_xy</i>	Point around which the polygon is to be rotated
<i>b_copy</i>	Optional, if t does the rotation on a copy. Default is nil.

### **Value Returned**

<i>lo_polygon</i> / <i>o_polygon</i>	In-place offset ( <i>g_copy</i> nil) of provided polygons or a rotated copy of the provided polygons ( <i>g_copy</i> is t).  If passed a list of polygons returns a list, otherwise returns a single polygon.
---	---

### **Example**

```
<cdsroot>/share pcb/examples/skill/axlcore/ashpoly.il
```

### **See Also**

[axlPolyFromDB](#), [axlPolyOffset](#)

## axlPolyTrim

```
axlPolyTrim  
  o_polygon1/lo_polygon1  
  s_trimMode  
  [f_cornerRadius]  
==> lo_polygon/nil
```

### Description

This function performs smooth/trim type adjustments to the provided polygons to ensure that items like acute angles are properly cleaned up and ready for artwork generation.

*s\_trimMode* values match those in the Global shape parameters:

- 'CHAMFER: Trim with chamfer style settings.
- 'ROUND: Trim with round style settings.
- 'FULL\_ROUND: Trim with full-round settings.

### Arguments

<i>o_polygon1</i> / <i>lo_polygon1</i>	Polygons to process.
<i>s_trimMode</i>	Symbol specifying the trim style to perform.
<i>f_cornerRadius</i>	Corner radius when performing round/full-round trim

### Value Returned

<i>lo_polygon</i>	Returns a list of <i>o_polygons</i> which represent the resulting geometry after performing given trims
<i>nil</i>	In case of error

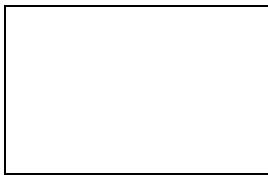
## Use Models

### Example 1

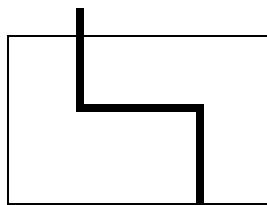
The existing Split Plane functionality can use the AXL version of the logical operation.

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.

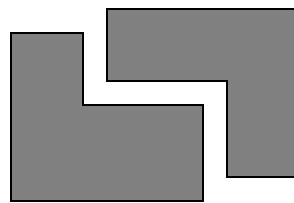
### Split Plane Usage



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
```

## Allegro SKILL Reference

### Polygon Operation Functions

---

```
(; 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 SKILL Reference**

### Polygon Operation Functions

---

---

## Allegro PCB Editor File Access Functions

---

### AXL-SKILL File Access Functions

This chapter describes the AXL-SKILL functions that open and close Allegro PCB Editor files.



Use these functions, instead of SKILL's infile and outfile, to access files using Allegro 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.

### **See Also**

[axlDMOpenFile](#)

### **Example**

```
q = axlDMOpenFile("TEMP" "foo.bar" "r")  
unless(q  
printf("ERROR is %L\n" axlDMFileError()))
```

## **axlDMFindFile**

```
axlDMFindFile (
    t_id
    t_name
    t_mode
    [t_prop]
)
⇒ t_name/nil
```

### **Description**

Opens a file using Allegro PCB Editor conventions. Adds an extension and optionally looks it up in an Allegro 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: <code>r</code> : read-only <code>w</code> : write <code>wf</code> : write line-buffered
<i>t_prop</i>	Property string.

### **Value Returned**

<i>t_name</i>	Name of file opened.
<code>nil</code>	Failed to open file.

### **See Also**

[axlDMOpenFile](#)

## **Allegro SKILL Reference**

### Allegro PCB Editor File Access Functions

---

#### **Example**

```
(setq aPort(ax1DMFindFile "ALLEGRO_TEXT","clip","w",":HELP=clipboard"))
```

Finds the fully qualified name `clip.txt` for writing.

## **axlDMGetFile**

```
axlDMGetFile(  
    t_id  
    t_name  
    t_mode  
    [t_prop]  
)  
⇒ t_name/nil
```

### **Description**

Gets the file name *t\_name* using Allegro PCB Editor conventions as described in the arguments. Returns the full path name of the file. Displays an error message if the file cannot be opened.

## Arguments

<i>t_id</i>	File attribute id.
	This string must be one of the types in the Allegro PCB Editor system file fileops.txt. Examples are ALLEGRO_LAYOUT_DB for Allegro PCB Editor layouts with extension brd, ALLEGRO_REPORT for Allegro PCB Editor report files with extension rpt.
<i>t_name</i>	String giving name of the file to open.
<i>t_mode</i>	Open mode. One of the following: <i>r</i> : read-only <i>w</i> : write <i>wf</i> : write line-buffered
<i>t_prop</i>	Property string.

## 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.

## See Also

[ax1DMOpenFile](#)

## Example

```
myfile = ax1DMGetFile( "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

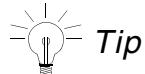
Opens a file in conventional Allegro manner; adds an extension and optionally looks it up in an Allegro search path. Must have an entry in `fileops.txt` file.

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 (the extension is appended 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.



Use `get_filename(p_port)` to obtain the name of the file.

## Arguments

<i>t_id</i>	File attribute id. This string must be one of the types in the Allegro PCB Editor system file <code>fileops.txt</code> . Examples are <code>ALLEGRO_LAYOUT_DB</code> for Allegro PCB Editor layouts with extension <code>brd</code> , <code>ALLEGRO_REPORT</code> for Allegro PCB Editor report files with extension <code>rpt</code> .
<i>t_name</i>	String giving the 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, create if doesn't exist, truncate to zero length if exists <code>a</code> : 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 you 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, this has the effect for reading of not eliminating the carriage-returns (`\r`) that are in DOS ASCII files. For writing, it does not add the carriage-returns when it sees a linefeed (writes it like a UNIX ASCII file). Example "`rb`" .

`s`: Allow spaces in the file or directory name. Currently, Allegro does not support this behavior. Setting this option is unsupported. Example "`rbs`" .

## Value Returned

<i>p_port</i>	Port of the opened file. In the case of <code>t_mode = "r"</code> , the file must exist for successful completion.
<code>nil</code>	File not found. Displays a confirmmer giving the name of the file it could not find.

## See Also

[ax1DMFileError](#), [ax1DMFindFile](#), [ax1DMGetFile](#), [ax1DMOpenLog](#),  
[ax1DMClose](#), [ax1DMFileParts](#)

## Example

Opens a file `clip.txt` for writing.

```
aPort = axlDMOpenFile("ALLEGRO_TEXT" "clip" "w")
```

Opens a file `b.bar`.

```
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 confirmmer giving the name of the file it could not find.

### **See Also**

[axlDMDOpenFile](#)

### **Example**

```
logport = axlDMDOpenLog("clipboard")
```

Opens the file clipboard.log for writing.

## **axlDMClose**

```
axlDMClose(  
    p_port  
)  
⇒ t/nil
```

### **Description**

Closes a file currently open in Allegro PCB Editor. Instead of using SKILL's infile/outfile commands, this command must be used to close the files opened using `axlDMOpenFile` or `axlDMOpenLog` commands.



While the core SKILL function, `close` can be used to close a file, it is recommended that any file opened using an `axlDM` function, must be closed using this function. Programs that adhere to this standard, will be compatible with future Allegro Data Management enhancements.

### **Arguments**

*p\_port*                   Id of the open port to be closed.

### **Value Returned**

<code>t</code>	Closed the file.
<code>nil</code>	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 Allegro 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

<i>t_adsFileType</i>	First entry in fileops.txt.
<i>t_title</i>	Title for the dialog
<i>t_helpTag</i>	Tag for the help file (used only by Cadence)

### Value Returned

<i>t_filename</i>	Full path to the filename.
nil	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 ( <code>t</code> ), or for read ( <code>nil</code> ).
<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.
<code>nil</code>	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

<i>t_id</i>	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.
<i>g_writeFlag</i>	If the file is to be opened for write ( <code>t</code> ), for read ( <code>nil</code> ).
<i>t_defaultName</i>	Name of file to select by default.
<i>t_helpTag</i>	Tag that defines the help message to display if the Help button is selected in the browser. Default help provided if option not set.
<i>g_directorySet</i>	Sets the directory change button which, by default, is not set.
<i>g_noDirectoryButton</i>	Hiding of the directory change button in the browser. By default, the button is present.
<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 <code>t</code> , then it starts the browser in the current working directory. Normally, you should set this option if <code>g_directorySet</code> is <code>t</code> .
<i>g_mainFile</i>	Matches options Allegro PCB Editor uses to open files from the File menu. This is <code>g_noSticky=t &amp; g_directorySet=t</code> . 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 <code>t_id</code> filter. The format is: <code>&lt;msg&gt; &lt;filter&gt; &lt;msg&gt; &lt;filter&gt;...</code>

## Value Returned

<i>t_fileName</i>	Name of the file selected.
<code>nil</code>	No file selected.

## Examples

- Browses Allegro PCB Editor text files.

```
axlDMFileBrowse ("ALLEGRO_TEXT" nil)
```

## **Allegro SKILL Reference**

### Allegro PCB Editor File Access Functions

---

- Browses Allegro PCB Editor text files and allows secondary filter of \*.log.

```
axlDMFileBrowse("ALLEGRO_TEXT" nil ?optFilters "All log files (*.log)|*.log")
```

- Browse SKILL files (both .il and .ils extensions).

```
axlDMFileBrowse(nil nil ?optFilters "Skill files (*.il)|*.il|Skill  
Oops (*.ils)|*.ils")
```

## **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*)

### **See Also**

[axlDMOpenFile](#)

### **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 (t/nil)

### **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") )
```

## **axIOSFileMove**

```
axIOSFileMove (
    t_src
    t_dest
)
⇒ t/nil
```

### **Description**

Moves the given source file to the given destination.

### **Arguments**

<i>t_src</i>	Full path of the source file.
<i>t_dest</i>	Full path of the destination file.

### **Value Returned**

<i>t</i>	Moved file.
<i>nil</i>	Failed to move file.

### **Example**

```
unless (axIOSFileMove("/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**

<i>t_directory</i>	Directory path using UNIX style slashes (/).
nil	Failed due to incorrect argument.

### **See Also**

[axlOSBackSlash](#)

### **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.



#### Caution

***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

<i>t</i>	Directory is successfully removed.
<i>nil</i>	Failed for one of the following reasons: <ul style="list-style-type: none"><li>- doesn't exist</li><li>- read protected</li><li>- sub-file or directory does not allow remove (<i>partial success</i>)</li><li>- sub-file or directory is in use (NT only) (<i>partial success</i>)</li></ul> A partial success means that some of the files and directories were deleted.

## **Allegro SKILL Reference**

### Allegro PCB Editor File Access Functions

---

#### **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 SKILL Reference**  
Allegro 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>	List of file names for the extract output. If one file is specified, output is string instead of list. Maximum of 24 files are supported. If more than 24 views are required, use multiple view files.
<i>lt_options</i>	<p>List of keyword strings for various options:</p> <ul style="list-style-type: none"><li>■ "crosshatch": Generate crosshatch lines when extracting crosshatched shapes.</li><li>■ "ok": Unknown field names are OK. Useful when the extract command file has property names not defined for the design being extracted.</li><li>■ "short": Extract data in the short format</li><li>■ "quiet": Do not print progress messages on <code>stdout</code>.</li><li>■ "mcm": Output MCM terminology.</li></ul>

## 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

```
axlExtractToFile( "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.
nil	Failed to apply SKILL functions to specified objects. See the file extract.log 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
```

## **axlReportGenerate**

```
axlReportGenerate(  
    t_reportName  
    g_showReport  
    [t_fileName]  
) -> t/nil  
  
axlReportGenerate(  
    'list  
    nil  
) -> t
```

### **Description**

This generates a report from one of the Allegro available reports.

For development purposes, if the report name is 'list a file, `reports_list.txt`, is created with the names of all supported reports.

Many, but not all, of these reports are available at the OS level via the `report` command.

### **Cautions**

- This only supports the reports available via the `reports` command. Allegro has application generated reports that cannot be accessed via this interface.
- Reports may be added or deleted and their names may change across releases.
- Report formats may change. You may need to update the code for report enhancements.

### **Arguments**

<i>t_reportName</i>	Name of report or 'list if you want a list of reports
<i>g_showReport</i>	If <i>t</i> , show the report to the user
<i>t_fileName</i>	Save report to indicated name

### **Value Returned**

<i>t</i>	Successful
<i>nil</i>	Problem with the arguments, the interface does not know if the report generation was successful

## See Also

[axlReportRegister](#)

## Examples

- Generate a report and save to file

```
axlReportGenerate("Bill of Material Report" nil "bom.txt")
```

- See what reports are available

```
axlReportGenerate('list nil)
```

## **axlReportList**

```
axlReportRegister(  
    ) -> ll_reportList/nil
```

### **Description**

Lists all the SKILL reports registered to the Allegro PCB Editor report interface. Even if you do not register any reports, Allegro 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/nil
```

### Description

Allows registration of user reports using the Allegro PCB Editor Reports dialog box. You provide a report name (displayed in the dialog), title (displayed in the title bar when the report is shown), and a report-generating callback function.

Callback is of format where a filename is passed that you should open, write your report and close. Callback function format is: `g_reportCallback(t_outfile)` .

The callback should, if it generates a report, return a `t` from the function. If it does not generate a report return a `nil`.

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:

## Allegro SKILL Reference

### Reports and Extract Functions

---

```
<a href="xprobe:xy:<x>,<y>">(x y)</a>
```

Example: xy coordinate is (310 140)

```
<a href="xprobe:xy:310.0,140.0">(310.0 140.0)</a>
```

#### Tips 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. You cannot ask for picks from the Allegro canvas. 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>))
- The provided SKILL callback function must return a
  - 't' if you want to the report to display.
  - A 'nil' if there is no report to display.
- 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 autoload file (.al) if building autoload contexts. This is the same rule that applies to axlCmdRegister.

## Allegro SKILL Reference

### Reports and Extract Functions

---

#### Arguments

<i>g_reportCallback</i>	Symbol or string of a SKILL function.
<i>t_name</i>	Name of report (shown in reports dialog box); if nil will delete the report associated with <i>g_reportCallback</i> .
<i>t_title</i>	Name in title bar when displaying report.

#### Value Returned

<i>t</i>	Arguments correct. A report is registered.
nil	An 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.

```
axlReportRegister('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 (reportName "helloWorld"))  
  
    let( (fp)  
        fp = axlDMOpenFile("ALLEGRO_REPORT" reportName "w")  
        axlLogHeader(fp "This is my report")  
        fprintf(fp, "\nHello World\n")  
        close(fp)  
        t  
    ))
```

- Unregister above report:

```
axlReportRegister('MyReport, nil nil)
```

#### See Also

[axlReportList](#) [axlLogHeader](#)

# **Allegro SKILL Reference**

## Reports and Extract Functions

---

# **Allegro SKILL Reference**

## Reports and Extract Functions

---

---

## **Utility Functions**

---

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.

## axlCheckString

```
axlCheckString(  
    t_type  
    t_string  
) -> t_modString/nil  
  
axlCheckString(  
    nil  
    nil  
) -> lt_types  
  
axlCheckString(  
    'error  
    nil  
) -> t_errorMsg/nil
```

### Description

Checks the provided string for legal characters and length. You must also provide the data type. Give this API two nils if you want to see the list of supported data types.

It performs the following checks for the data type provided:

- length check
- legal characters

For all data types except PROPVALUE, it may also modify the provided string and return the updated string to meet Allegro database rules.

- stripping leading and trailing white space
- upper case

Unique considerations by data type:

- PROPVALUE checks for common property value restrictions. Typically, a property may have more severe restrictions.
- GROUP checks for common group naming restrictions. A particular group may enforce additional restrictions.

You should use the return string since it may have been modified.

Errors messages may evolve over time. Last error is only maintained until next call to axlCheckString.

## Arguments

<i>t_type</i>	data type (nil to return all supported data types)
<i>t_string</i>	input string
'error	fetch last error message

## Value Returned

<i>t_modString</i>	Modified output string if check is successful
<i>lt_types</i>	List of data types supported
<i>nil</i>	Failed check

## See Also

[axlDBControl](#) - max name length

## Examples

- return all supported types

```
axlCheckString(nil nil)
```

- typical check

```
axlCheckString("REFDES" "u1") -> "U1"
```

- error

```
axlCheckString("NET" "!GND") -> nil
```

```
axlCheckString('error nil)
-> "ERROR(SPMHUT-1): Illegal character(s) present in the name or value."
```

## 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 Allegro 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.

Enable debug also enables error messages for invalid attributes of Allegro *dbids*. This enables detecting typos when fetching data from *dbids*. This entails a slight performance hit. For example, if you have a pin *dbid* and you do a pin->bbox (instead of pin->bBox), then an error will be issued.

### Arguments

#### Values Returned

t	To enable AXL SKILL debug mode.
nil	To disable.

Returns last setting

### See Also

[axlIsDebug](#)

## **axlDetailLoad**

```
axlDetailLoad(  
    t_filename  
    point  
    f_scale  
    x_rotation  
    g_mirror)  
==> t/nil
```

### **Description**

This loads a the designated `ipf` file (`t_filename`) into the current design at location (`point`), with scaling (`f_scale`), rotation (`x_rotation`) and mirror (`g_mirror`). Items are clipped to rectangle provided a save file time. It will load the detail to the active layer.

**Note:** scale values much less then 1.0 may have unpredictable results.

### **Arguments**

<code>t_filename</code>	the name of the plot file which contains the detail.
<code>point</code>	the location to place the detail.
<code>f_scale</code>	the scaling factor of the details as a floating point number (1.0 is no scaling). Must be greater then 0.1
<code>x_rotation</code>	the rotating angle in degrees. Rotation is restricted to integers.
<code>g_mirror</code>	whether the detail should be mirrored (t/nil).

### **Value Returned**

<code>t</code>	was able to load
<code>nil</code>	otherwise

### **Examples**

Select a group of objects and load at 0,0 with double scaling.

- use `ashSelect` function in SKILL examples area

```
file = "myplt.plt"
```

## Allegro SKILL Reference

### Utility Functions

---

```
objs = ashSelect(nil)
lyr = "MANUFACTURING/DETAIL"
win = axlDBGetDesign()->bBox
when(objs
    axlDetailSave(file win objs)
    axlLayerCreateNonConductor(layer)
    axlVisibleLayer(layer t)
    axlSetActiveLayer(layer)
    axlDetailLoad(file 0:0 2.0 0 nil)
)
```

#### See Also

[axlDetailSave](#), [axlSetActiveLayer – Obsolete](#)

## **axlDetailSave**

```
axlDetailSave(  
    t_filename  
    l_bBox  
    o_dbid/lo_dbid  
    [g_filledPads]  
)  
==> t/nil
```

### **Description**

This saves a clipping box (*l\_bBox*) and the passed set of geometries (*lo\_dbid*) to a Allegro ipf file (*t\_filename*).

### **Notes:**

- Any attached text to the dbids is saved.
- Ignores logical dbids such as nets, components, etc.

### **Arguments**

<i>t_filename</i>	the name of the file into which the detail is saved
<i>l_bBox</i>	list of two xy coordinates defining selection box
<i>lo_dbid</i>	list of AXL DBID's or single DBID
<i>g_filledPads</i>	output filled pads as filled (t) or unfilled (nil). Default is filled. Certain other filled figures may be effected by this option

### **Value Returned**

<i>t</i>	was able to save into the file
<i>nil</i>	otherwise

### **See Also**

[axlDetailLoad](#)

## 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; nil 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.
nil	Failure.

### Example

```
axlEmail("aperson" nil "A test message" "anybody home?")
```

## axlHistory

- Report history buffer (first to last)

```
axlHistory(  
    [x_num]  
) -> t
```

- Read or write history to a file

```
axlHistory(  
    s_operation  
    t_filename  
) -> t/nil
```

## Description



***This is a developers aid only. Do NOT use it in a SKILL program.***

Provides command recall capability to the SKILL development window.

Functionality also applies to the Allegro PCB Editor command line except the command is: "history <n>" where n is the print the last N commands.

The command line has no ability to read or write history files. They are automatically read on program startup and saved on exit.

History environment variables:

- allegro\_history = <n> (default is 200 commands)

Command recall buffer length.

- allegro\_savehist = <n> (default off)

Save history file to be saved on program exit. Will be read next time on startup. File is saved at <HOME>/pcbenv/history\_<name>.txt

where <name>

- is SKILL for the SKILL command area
- program name for Allegro command area

History support:

- !! – last command (same as !-1)

## Allegro SKILL Reference

### Utility Functions

---

- !<num> – redo command number (ex ! 5)
- !-<num> – redo relative to last cmd (ex !-2)
- !<str> – redo cmd starting with string last to first search (ex !echo)
- !?<str> – redo cmd matching with string last to first search. For example, !?unnamed

### Arguments

<i>x_num</i>	print last <num> commands, 0 or no argument prints entire recall buffer
<i>s_operation</i>	'read – read history buffer from provided file and append to history 'write – write history buffer to provided file
<i>t_filename</i>	A history file (default extension is .txt)

### Value Returned

<i>t</i>	Operation succeeded
<i>nil</i>	Failed

Also prints history buffer if running with option 1.

### Example

- Report history buffer

```
a = 1
b = 2
axlHistory()
```

- Save history

```
axlHistory('write "skillhist")
```

- Read history

```
axlHistory()
axlHistory('read "skillhist")
axlHistory()
```

## **axlHttp**

```
axlHttp(  
    t_url  
)  
⇒ t
```

### **Description**

Displays a URL from Allegro 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.

## **Allegro SKILL Reference**

### Utility Functions

---

#### **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**

```
a xlHttp("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") )
```

## **axlIsProductLineActive**

`axlIsProductLineActive(t_productLine) -> t/nil`

### **Description**

This routine determines if a product in a given product line has been started.

### **Arguments**

`productLine`      The product line that you want to check.

The legal values are:

- SI
- PCB
- CONCEPT
- ORCAD
- APD

### **Value Returned**

- `t`: There is a license checked out for the given product line.
- `nil`: There is no license checked out for the given product line.

## **axlIsProductStarted**

```
axlIsPointInsideBox(  
    t_productName  
) -> t/nil
```

### **Description**



*Caution*

***This should not be used, use axlLicIsProductEnabled.***

## **axILicDefaultVersion**

```
axILicDefaultVersion(  
    ) -> f_version
```

### **Description**

This returns the default version number used in licensing. Most applications in a release will use this version number when checking out licenses.

### **Arguments**

None

### **Value Returned**

f\_version - floating point number (forexample,16.6)

## **axILicFeatureExists**

```
axILicFeatureExists(  
    t_license  
    [f_version]  
) -> t/nil
```

### **Description**

Checks if license feature exists. Does not verify that the license is available. If no version is provided, the tool default version (see [axILicDefaultVersion](#)) is used.

### **Argument**

<i>t_license</i>	license name
<i>f_version</i>	floating point number (forexample, 16.3)

### **Value Returned**

*t* if checkout, *nil* in case of failure

### **See Also**

[axILicIsProductEnabled](#)

## **axILicIsProductEnabled**

```
axILicIsProductEnabled(  
    t_license  
) -> t/nil  
  
axILicIsProductEnabled(  
    `all  
) -> lt_licenses
```

### **Description**

Checks if license is checked-out by tool.

In first form, if given a license string returns `t` when license is checked-out and `nil` if it isn't. The test is based on the result obtained by running tool. There is no SKILL method to determine licenses.

In the second form, if called with the symbol `all` returns a list of licenses currently checked out by the tool.

### **Arguments**

<code>t_license</code>	Name of license (case sensitive)
<code>'all</code>	Get all licenses currently checked-out by the program.

### **Value Returned**

- `t` - license checked out
- `nil` - license not checked out
- `lt_licenses` - list of licenses checked-out by program (all mode)

### **See Also**

[axILicFeatureExists](#), [axILicDefaultVersion](#)

### **Examples**

See if SKILL developer license (product 900) is checked out:

```
axILicIsProductEnabled("skillDev")
```

## axlLogHeader

```
axlLogHeader(  
    p_port  
    t_titleString  
    [t_prefix]  
)  
⇒ t/nil
```

### Description

Writes the standard Allegro PCB Editor log file header to the passed open file. The header record includes:

- Title String
- Drawing Name
- Software Release
- Date and Time

### Arguments

<i>p_port</i>	SKILL port for the open file.
<i>t_titleString</i>	Title string in the log file header.
<i>t_prefix</i>	Optional prefix string to header for easier parsing (recommend "#")

### 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 Allegro 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.

## **Allegro SKILL Reference**

### Utility Functions

---

#### **Example 1**

```
ax1MKS2UU("100.1") -> 100.0
```

Default conversion with database in mils.

#### **Example 2**

```
ax1MKS2UU("100 mils") -> 100.0
```

Database is in mils.

#### **Example 3**

```
ax1MKS2UU("100 inches") -> 100000.0
```

Database is in mils.

#### **Example 4**

```
ax1MKS2UU("100 METER") -> 3937008.0
```

Database is in mils.

## axlMKSAlias

```
axlMKSAlias(  
    t_MKS Alias  
)  
⇒ 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)  
axlMKSAliaas("M") -> "METER"  
axlMKSAlias("KG") -> NIL (The function supports only the use of basic units.)
```

## **axlMKSCConvert**

Operates in several ways, depending on the arguments passed.

1. To convert a number from an input units type to an output unit type, general case:

```
axlMKSCConvert (
    n_input
    t_inUnits
    [t_outUnits]
)
=> f_output/nil
```

2. To convert a number from an input units type to MKS units:

```
axlMKSCConvert (
    n_input
    t_inUnits
    nil
)
=> f_output/nil
```

3. To convert a number from MKS units to an output unit type:

```
axlMKSCConvert (
    n_input
    nil
    t_outUnits
)
=> f_output/nil
```

4. To convert an MKS string to an output units, i.e. ".5 MILS" to "INCHES":

```
axlMKSCConvert (
    t_input
    [t_outUnits]
)
=> f_output/nil
```

5. To pre-register an input units, so that subsequent calls need not specify units:

```
axlMKSCConvert (
    nil
    t_inUnits
)
=> t/nil
```

6. To convert a unit from a pre-registered input units:

```
axlMKSCConvert (
    n_input
)
=> f_output/nil
```

## **Description**

Converts any allowable unit to any other allowable unit. It operates in several ways, depending on the arguments.

## **Allegro SKILL Reference**

### Utility Functions

---

In all instances with `t_outUnits` as an argument, if that argument is omitted, the function converts to active design units.

1. If the first argument is a number and the second argument is evaluated as the units of that number, the number is converted to the units specified by the third argument.

A special case is if first argument is a number, the second is "design" and the third is a length unit then the input number is converted from current design units to specified length units.

2. In the case where the first argument is a string, that string must be a fully instantiated units string, i.e. "15 MILS". In this case the second string is interpreted as the output units, and the value of the first string is converted to these units.
3. If the first argument is nil, then the input units specified by the second argument is remembered for the future calls. The output units will default to the active design, and the function will fail as above if no design is active. Note that pre-registration will not work if there is no active design.
4. When only a number is specified, the function will confirm that an input unit has been pre-registered, and will convert that number, in the pre- registered units, to the design units. Function will issue a warning and return nil if no pre-registered input units, or no active design
5. When the second and third arguments are present but one is nil, it represents MKS standard units for the input or output units, respectively.

## Arguments

<i>n_input</i>	Number to convert
<i>t_inUnits</i>	String giving the units of the input number. If first argument is not specified, this string should specify the input number to convert and its units.
<i>t_outUnits</i>	String giving the new units for <i>f_output</i> . If <i>t_outUnits</i> is nil, converts the number to the current units of the layout.

## Value Returned

<i>f_output</i>	Converted value of <i>n_input</i> .
nil	Failed to convert value due to incorrect arguments.

## Examples

- Typical use, convert from design units to another type (design is in mils)

```
axlMKSCConvert(.5 "design" "INCHES") => 0.0005  
axlMKSCConvert("5 MILS" "INCHES") => 0.0005
```

- Pre-register a unit type then use it on future conversions

```
axlMKSCConvert(nil "MILS") => t  
axlMKSCConvert(.5) => 0.0005
```

- Go to default output conversion (METERS)

```
axlMKSCConvert(.5 "MILS" nil) => 2.54e-05
```

- Not just length is supported (go to farads)

```
axlMKSCConvert(1e-09 nil "pF") => 1000.0
```

## 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

<i>t_mksString</i>	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.

## Allegro SKILL Reference

### Utility Functions

---

#### axlMapClassName

```
axlMapClassName (
    t_oldName /! t_oldName
    [g_mapToPCB]
)
⇒ t_newName /! t_newName
```

#### Description

Use this function to write a SKILL program that runs in Allegro PCB Editor and APD. You can map from class names to the name appropriate to the program running your SKILL program. For example, you can write a program using Allegro PCB Editor class names and have the program run in APD.

The table shows the name mapping between Allegro PCB Editor and APD.

**Table 24-1 Class Name Mapping**

Allegro PCB Editor Class Name	APD Class Name
BOARD GEOMETRY	SUBSTRATE GEOMETRY
ETCH	CONDUCTOR
ANTI ETCH	ANTI CONDUCTOR
PACKAGE GEOMETRY	COMPONENT GEOMETRY
PACKAGE KEEPIN	COMPONENT KEEPIN
PACKAGE KEEPOUT	COMPONENT KEEPOUT
BOARD	SUBSTRATE

When using the list of names mode, names that are not recognized are returned "as is".

## Allegro SKILL Reference

### Utility Functions

---

#### Argument

<i>t_oldName</i>	Allegro PCB Editor class name.
<i>lt_oldName</i>	List of Allegro PCB Editor class name.
<i>g_mapToPCB</i>	Optional. <i>t</i> maps from the APD name to the PCB names. Default is <i>nil</i> (PCB to APD conversion).

#### Value Returned

<i>t_newName</i>	Appropriate class name based on product type.
<i>lt_newName</i>	List of class names renamed.

#### Examples

1. Get APD class name when in APD.

```
axlMapClassName ("ETCH") -> "CONDUCTOR"
```

2. Gets Allegro PCB Editor class name when in Allegro PCB Editor.

```
axlMapClassName ("ETCH") -> "ETCH"
```

3. List in APD

```
axlMapClassName (' ("ETCH" "ANALYSIS")) -> ("CONDUCTOR" "ANALYSIS")
```

## **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**

<i>x_size</i>	Estimate of memory use in bytes.
---------------	----------------------------------

## **axIOSBackSlash**

```
axIOSBackSlash(  
    t_directory  
)  
==> t_directory/nil
```

### **Description**

This changes UNIX style forward slashes to DOS style backslashes. While most applications support either style, certain older Windows applications only support DOS style path.

The UNIX style is more amenable to SKILL programming.

On UNIX this just returns the incoming string.

### **Arguments**

*t\_directory*      String containing directory path with forward slashes

### **Value Returned**

*t\_directory/nil*

### **See Also**

[axIOSSlash](#)

### **Example**

```
p = axIOSBackSlash("/tmp/foo")  
-> "\\tmp\\foo"
```

## Allegro SKILL Reference

### Utility Functions

---

#### axIOSControl

```
axIOSControl(
    s_name
    [g_value]
)
==> g_currentValue/ls_names
```

#### Description

Inquires and/or sets the value dealing with the graphics. If setting a value, the return is the old value of the control.

A side effect of most of these controls is if a form is active that is displaying the current setting it may not be updated. Additional side effects of individual controls, currently supported, are listed in the following table.

Name	Value	Set?	Description	Equivalent	Side Effects
cpu	x_number	No	Returns number of available CPUs. Included in the number are multi-cpu multi-core, and hyper threading.	none	none
is64exe	t/nil	No	Returns t if this executable is 64bit	none	none
is64os	t/nil	No	Returns t if this is a 64bit based OS.	none	none
isWindows	t/nil	No	Returns t if a Windows OS and nil if UNIX or Linux.	none	none
hostname	string	No	Returns hostname of computer this programming is running on.	none	none
physicalMemory	x_number	No	Returns in units of 1Mbyte the amount of physical memory. This is not to be confused with virtual memory.	none	none
			<b>Note:</b> 32bit OS can at most address 4GB (4000MB) of memory. 32bit Windows is restricted to 3GB. So even if you have more installed on your PC this will still report 3GB.		

## Arguments

<i>s_name</i>	Symbol name of control. If this value is <code>nil</code> , all possible names are returned.
<i>g_value</i>	Optional symbol value to set. Usually a <code>t</code> or a <code>nil</code> .

## Value Returned

See above

<i>ls_names</i>	If name is <code>nil</code> then returns a list of all controls.
-----------------	--

## Example

### 1. Get CPUs

```
size = axlOSControl('cpu)
-> 2
```

### 2. Get if 64bit program

```
axlOSControl('is64exe)
-> nil
```

## See Also

[axlUIControl](#), [axlMemSize](#)

## **axlOSNtp**

```
axlOSNtp(
    s_mode
    t_serverName/nil
)
==> x_nwtime/t_nwtime/t_server/nil
```

### **Description**

Reports time from a NTP server.

If the value of *s\_mode* is set to 'name, then Allegro PCB Editor reports the default NTP server, obtained using the environment variable NTPSERVER. Default is 0.pool.ntp.org.

Currently, the axlOSNtp command uses a timeout of 50 milliseconds.

On Linux, you can query NTP time using the following command.

```
/usr/sbin/ntpdate -q <hostname>
```

**Note:** IPv6 networks not supported.



***Must be connected to Internet and, if a firewall is enable, it must be configured to allow port 123.***

## Arguments

<i>s_mode</i>	The possible values are:  'name: Returns default NTP server name for Allegro PCB Editor.  t: Returns network time and date as string (current time zone)  nil: Returns time ID (seconds since 00:00:00 UTC, January 1, 1970)
<i>t_serverName</i>	Use NTP server or if the value specified in <i>nil</i> , default Allegro NTP server is used.

## Value Returned

- *x\_nwtime*: A time ID. This value is returned when the value of *s\_mode* is set to *nil*.
- *t\_nwtime*: Time and date in ASCII using current time zone. This value is returned when the value of *s\_mode* is set to *t*.
- *t\_server*: Name of the default time server. Returned when the value of *s\_mode* is set to 'name.
- *nil*: Returned in case of an error.

## Example

```
time = axlOSNtp(t nil)
```

## **axIoseExit**

```
axIoseExit(  
    x_return  
)  
⇒ no return
```

### **Description**

This exits the program to the OS with the return value provided. The database is not saved. Typically for Allegro programs:

0	Success
1	Warnings
2	Error

### **Arguments**

*x\_return*

### **Value Returned**

This does not return to the calling SKILL procedure.

### **Example**

```
axIoseExit(0)
```

## **axlPPrint**

```
axlPPrint(  
    t_name  
)  
⇒ t_pname
```

### **Description**

Converts a string with Allegro 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"
```

## **axlPdfView**

```
axlPdfView(  
    t_pdfFile  
)  
⇒ t
```

### **Description**

Displays a PDF file from Allegro PCB Editor. If just the filename is given, attempts to find the PDF file using Allegro 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**

<i>t</i>	PDF file displayed.
<i>nil</i>	Failed to display PDF file due one of the following: <ul style="list-style-type: none"><li>- No Acroread PDF viewer found on UNIX.</li><li>- No PDF viewer registered on Windows.</li></ul>

### **Example**

```
axlPdfView("allegro.pdf")
```

## **axlPrintDbid**

```
axlPrintDbid(  
    o_dbid/lo_dbid  
    [o_port]  
) -> t
```

### **Description**

This is a debug function to print one or a list of dbids. Output format is terse and parsable. This will not print all attribute data in a dbid but is customized to aid in understanding the dbid data.

Format: <key> <attribute> <value>

where key can be

- c common value of attribute (objType, name, xy)
- a other attributes
- l printable detail on list of dbids associated with element Not all list of dbids are reported.
- g what groups have the object.

Format:

```
g group <type> name= <name>
```

Output is either directed to the SKILL window or if port is provided written to a file.

Note using the environment variable, `pv_showelem` you can get the same data in the *show element* command.

### **Arguments**

<code>o_dbid/lo_dbid</code>	dbid or a list of dbids
<code>o_port</code>	option port object (from outfile)

### **Value Returned**

t

## **Allegro SKILL Reference**

### Utility Functions

---

#### **See Also**

[axlShowObject](#)

#### **Example**

```
l = axlGetDesign()  
axlPrintDbid(l)
```

# axlRegexpls

```
axlRegexpIs( t_exp ) ⇒ t/nil
```

## Description

Determines whether an environment variable expression contains Allegro 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 Allegro PCB Editor are more compatible with the character set allowed in the Allegro 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<sub>exp</sub>* SKILL symbol for the environment variable name.

## Value Returned

t	Expression contains Allegro PCB Editor compatible wildcards.
nil	Expression does not contain Allegro PCB Editor compatible wildcards.

## **axlRunBatchDBProgram**

```
axlRunBatchDBProgram(  
    t_prog  
    t_cmdFmt  
    [?logfile logfile]  
    [?startMsgt startMsg]  
    [?reloadDBt/nil]  
    [?noUnloadt/nil]  
    [?silentt/nil]  
    [?noProgress t/nil]  
    [?warnProgram 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.

The following options are always required (UIBatchSpawn):

- |                 |                             |
|-----------------|-----------------------------|
| <i>t_prog</i>   | Name of the program to run. |
| <i>t_cmdFmt</i> | Command string.             |

The following options are optional.

- |                  |   |
|------------------|---|
| <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 (for example, reports) then there is no reason to reload database. dbids are refreshed.               |

## Allegro SKILL Reference

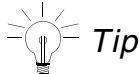
### Utility Functions

---

<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 Allegro PCB Editor database. Default is nil.
<i>silent</i>	If you set this to t, no messages are displayed. Use when the user does not need to know that a program is spawned. Default is nil.
<i>noProgress</i>	If t, the progress meter does not display. Default is nil.
<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>warnProgram</i>	Program supports warning status (returns 0, if success; 1, if warnings; and 2, if errors).
<i>noExitMsgs</i>	When set, suppresses messages about program success or failure.

**Note:** For *t\_cmdFmt*, the formatting should include everything except the design filename. Place a %s where the design should appear. To get a %s while doing a *sprintf*, use a "%s" as shown in following examples:

```
cmdFmt = "netrev -$ -q -r %s"  
sprintf(cmdFmt, "%s -$ %s %s %%s", prog, argq, argr)
```



- a.** For debug, set the env variable, *wait\_debug*, on the Allegro command line:

```
set wait_debug
```

or in SKILL

```
axlSetVariable("wait_debug" nil)
```

This echos the spawning arguments of your program. You can also use this variable to see how Allegro spawns programs from its dialogs. The "#T<num>.tmp" name seen in this output is the temporary save of the current design to disk and the use of the "%s" argument in your *t\_cmdFmt* statement.

- b.** For a list of Allegro programs and their command line arguments see the directory:

```
<cdsroot>/share/pcb/batchhelp
```

or run the batch program with the "-help" argument.

Example: *idf\_in -help*

## Allegro SKILL Reference

### Utility Functions

---

- c. Many Allegro batch programs support ' -\\$ ' as a standard argument. This prevents prompting for missing input arguments.

#### Argument

<i>t_prog</i>	string containing program name
<i>t_cmdFmt</i>	string containing starting arguments (including program)
<i>t_logfile</i>	optional string showing logfile
<i>t_startMsg</i>	optional string having start message
? <i>reloadDB</i>	Optional t/nil having database be reloaded after job completes
? <i>noUnload</i>	Optional t/nil stating database shouldn't be saved
? <i>silent</i>	Optional t/nil controlling info messages
? <i>noProgress</i>	Optional t/nil controlling progress meter

#### Value Returned

<i>t</i>	Batch job ran.
<i>x_error</i>	Error number that is program dependent on failure.

#### Examples

- Spawn genfeedformat which requires design to be saved to a temp file

*Program Args:*

- -\\$ - silent
- -b - name of design (required)
- %%s - because we sprintf the format before calling batch we need to escape the %s by perpending an extra %

*SKILL:*

```
sprintf(format "genfeedformat -\$ %%s", "-b")
;format= "genfeedformat -\$ -b %%s"
axlRunBatchDBProgram("genfeedformat" format
?logfile "genfeed")
```

Without sprintf

## Allegro SKILL Reference

### Utility Functions

---

```
axlRunBatchDBProgram("genfeedformat" "genfeedformat -$ -b %s" ?logfile  
"genfeed")
```

- Spawn 3rd party program, notepad, on an existing file "allegro.jrl". In this case we do not want to save design (?noUnload t) and no progress meter is required (?noProgress t)

```
axlRunBatchDBProgram("notepad" "notepad allegro.jrl"  
?noUnload t ?noProgress t)
```

- Spawn 3rd party import login, a program that requires read/write database wrapping. Read existing 3rd party netlist file called "netlist.txt". Since design needs to be reloaded if import is successful use the "?reloadDB t" option.

#### *Program Args:*

- -\\$ - silent
- -g - run gate assign
- -y 1 - Always Place changed component
- netlist - name of netlist file (for example, netlist.txt)
- %s - use current design

#### *SKILL:*

```
axlRunBatchDBProgram("netin"  
"netin -$ -g -y 1 netlist %s"  
?startMsg "Logic Import"  
?logfile "netin"  
?reloadDB t)
```

- Export IDF

#### *Program Args:*

- -d IDF - File name type
- -V 3.0 - IDF version
- -h 2000 - default package height
- -s ... - Source id (note \"...\" allows spaces in name)
- -o myidf - output idf files (with bdf and ldf extensions)
- %s - axl will enter name of temp database saved to disk

#### *SKILL:*

```
axlRunBatchDBProgram("idf_out"  
"idf_out %s -d IDF -o myidf -s \"allegro_16.3\" -b 1 -h 2000 -v 3.0")
```

## **Allegro SKILL Reference**

### Utility Functions

---

- Import IDF, assumes an existing bdf file, unnamed.bdf. If successful load updated design back into memory via the reload option (?reloadDB t).

*Program Args:*

- -o %s - name of current design (%s - substitute in current design)
- unnamed - name of bdf file on disk

*SKILL:*

```
axlRunBatchDBProgram("idf_in" "idf_in -o %s unnamed" ?reloadDB t)
```

## 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 <b>Show Element</b> 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`.

## **axlSleep**

```
axlSleep(  
    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 axlEnterEvent.

If you are using the IPC interfaces then you should call `axlSleep` instead of using `sleep`.

### **Arguments**

<i>x_time</i>	Time in seconds.
---------------	------------------

### **Value Returned**

<i>t</i>	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.

## Allegro SKILL Reference

### Utility Functions

---

#### 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:  <i>field_number sortOrder fieldType</i>
	String can contain multiple triplets separated by commas. Triplets can contain valid elements as shown:

---

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.

<i>t_sort_options</i>	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:
-----------------------	--

## Allegro SKILL Reference

### Utility Functions

---

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 `alphlessp` 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.

<code>alphlessp sort</code>	<code>U1 U10 U2</code>
<code>axlstrcmpAlpNum sort</code>	<code>U1 U2 U10</code>

### **Arguments**

<code>t_str1</code>	A string
<code>t_str2</code>	A string

### **Value Returned**

<code>0</code>	The two strings are equal.
<code>+num</code>	If <code>t_str1</code> is greater than <code>t_str2</code> (1 goes after 2)
<code>nil</code>	If <code>t_str1</code> is less than <code>t_str2</code> (1 goes before 2)

### **Example**

```
l = '("U5" "U10" "U1" "U5" "U2")  
sort(l `axlstrcmpAlpNum)  
====> ("U1" "U2" "U5" "U5" "U10")
```

## axlStringCSVParse

```
axlStringCSVParse(  
    t_string  
    [g_stripWhite]  
    [t_separator]  
) -> lt_string/nil
```

### Description

Parses a comma delimited line (typical from Excel). This differs from the SKILL `parseString` function in several areas which make it compatible with Excel csv file format:

- `parseString` ignores two adjacent commas.
- if a separator is contained within a cell, Excel quotes the cell. `parseString` treats this as two cells.
- Correctly processes double quotes in a cell.

### Arguments

<i>t_string</i>	A csv string
<i>g_stripWhite</i>	Option to strip leading and trailing white space from each output string. Default is to leave whitespace. Do not advise using this option with separator set to " ".
<i>t_separator</i>	Optional separator character, default is a comma (,).

### Value Returned

<i>lt_string</i>	list of strings parsed based on comma separator.
<i>nil</i>	error (unlikely to happen)

### See Also

`parseString` in *Cadence SKILL Language Reference*

### Examples

- White space strip example

## Allegro SKILL Reference

### Utility Functions

---

```
ret = axlStringCSVParse(" a ,b" t)
-> ( "a" "b")
```

#### ■ Properly parse a csv file

```
ret = axlStringCSVParse("a,,c,\"d,e\"")
-> ( "a" "" "c" "d,e")
```

#### ■ Properly parse a csv file

```
ret = axlStringCSVParse("a||c|\\"d|e\"" nil "|")
```

## **axlStringRemoveSpaces**

```
axlStringRemoveSpaces(  
    t_string  
) -> t_modString/nil  
  
axlStringRemoveSpaces(  
    lt_string  
) -> lt_modString/nil
```

### **Description**

This will strip leading or trailing whitespace from a string (standard C `isspace()` macro). Has two modes:

- one string
- list of strings

In list of strings, items in list that are not a string are filtered out of the return.

### **Argument**

<i>t_string</i>	A string.
<i>lt_string</i>	list of strings

### **Value Returned**

<i>t_modString</i>	Modified string
<i>lt_modString</i>	Modified strings
nil	Not a string

### **See Also**

[axlCheckString](#)

### **Example**

```
ret = axlStringRemoveSpaces(" a ")  
ret = axlStringRemoveSpaces(' (" a " " b ")')
```

## axlVersion

```
axlVersion(  
    s_option  
)  
⇒ g_value/nil
```

### Description

Returns Allegro PCB Editor or OS dependent data.

### Argument

*s\_option*      SKILL symbol for the environment variable name.

### Value Returned

Return depends upon option given.

Option	Value	Returns
none	<i>ls_values</i>	List of available options.
version	<i>f_value</i>	Allegro PCB Editor program version, for example, 15.7.
tVersion	<i>t_value</i>	Allegro PCB Editor program version as a string, for example, 16.3.
fullVersion	<i>t_value</i>	Allegro PCB Editor program version and patch as a string (for example, 15.7 s01).
buildDate	<i>t_value</i>	Date program was built at Cadence (month/day/year), for example, 7/19/2006.

## Allegro SKILL Reference

### Utility Functions

---

Option	Value	Returns
release	<i>t_value</i>	Allegro 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 23 <sup>rd</sup> patch release.
internalVersion	<i>t_value</i>	Internal program version, for example, v13-5-26a.
programName	<i>t_value</i>	Executable being run, for example, allegro.
displayName	<i>t_value</i>	Short display name of program (may contain spaces.) Certain programs may change this during run-time depending on the license level. This may be the same or shorter than the formal name. It may change from release to release.  <i>Example:</i> "Allegro PCB Designer"
formalName	<i>t_value</i>	Long display name of program (may contain spaces). This is the full formal name of program and may change during run-time since it depends upon the license 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.
isAPDSi	<i>g_value</i>	<i>t</i> if Allegro Package Designer SI L , <i>nil</i> otherwise

## Allegro SKILL Reference

### Utility Functions

---

<b>Option</b>	<b>Value</b>	<b>Returns</b>
isAPDSiL	<i>g_value</i>	t if Allegro Package Designer SI L, nil otherwise
isAllegroExpert	<i>g_value</i>	t if Allegro Expert product, nil otherwise.
isAllegroDesigner	<i>g_value</i>	t if Allegro Designer product, nil otherwise.
isAllegroOrcad	<i>g_value</i>	t if Allegro OrCAD product, nil otherwise.
isAllegroPCB	<i>g_value</i>	t if Allegro PCB product, nil otherwise.
isAPD	<i>g_value</i>	t if Allegro Package Designer product, nil otherwise.
isAPDL	<i>g_value</i>	t if Allegro Package Designer L, nil otherwise
isAPDXL	<i>g_value</i>	t if Allegro Package Designer XL, nil otherwise
isAPDLegacy	<i>g_value</i>	t if Allegro Package Designer VT2300, nil otherwise
isChipIOPlanner	<i>g_value</i>	t if any Chip I/O Planner product, nil otherwise
isICP	<i>g_value</i>	t if any ICP based product (apd, sip), nil otherwise
isSIP	<i>g_value</i>	t if any SIP product, nil otherwise
isSIPDigital	<i>g_value</i>	t if any SIP Digital product, nil otherwise
isSIPDigArch_L	<i>g_value</i>	t if any SIP Digital Architect L product, nil otherwise
isSIPRF	<i>g_value</i>	t if any SIP RF product, nil otherwise
isSIPRFArch_L	<i>g_value</i>	t if any SIP RF Architect L product, nil otherwise
isSigxp	<i>g_value</i>	t if any version of SigXP product, nil otherwise.
isSxExpert	<i>g_value</i>	t if SigXP Expert product, nil otherwise.
isSxExplorer	<i>g_value</i>	t if SigXP Explorer product, nil otherwise.
isEmbedded	<i>g_value</i>	t if Embedded placement allowed.
isBackdrill	<i>g_value</i>	t if Backdrill functionality allowed.

## Allegro SKILL Reference

### Utility Functions

---

Option	Value	Returns
isToolbox	<i>g_value</i>	nil no toolbox, 'orcad – OrCAD toolbox active, 'allegro – Allegro toolbox  axlIsToolbox(): API is provided for legacy purposes and returns the same results.
isSymphony	<i>g_value</i>	t if running in Symphony (multi-user) mode.

## Examples

```
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 Allegro PCB Editor date/time format.
<i>nil</i>	Failed to print VERSION_ID due to incorrect argument.

### **Example**

```
axlVersionIdPrint(axlDBGetDesign () ->prop->VERSION_ID  
-> "Mon Dec 16 12:45:16 2004"
```

## **Allegro SKILL Reference**

### Utility Functions

---

## **Allegro SKILL Reference**

### Utility Functions

---

---

# Math Utility Functions

---

## Overview

This chapter describes the AXL-SKILL Math Utility functions.

### **axlDegToRad**

```
axlDegToRad(  
    n_angle  
) => f_angle
```

#### Description

Converts an angle in degrees to radians.

#### Arguments

*n\_angle*                    Angle in degrees

#### Value Returned

*f\_angle*                    Angle in radians

#### See Also

[axlRadToDeg](#), [axlMathConstants](#)

#### Example

```
axlDegToRad(45.0)  
=> 0.7853982
```

## **axlDistance**

```
axlDistance(  
    l_point1  
    l_point2  
)  
⇒ f_distance
```

### **Description**

Returns the distance between two points. You may use floating point.

### **Arguments**

<i>l_point</i>	A point.
<i>l1_line</i>	Two points at the ends of a line.

### **Value Returned**

<i>f_distance</i>	Distance between two points in floating point form.
-------------------	---

### **Example**

```
axlDistance(10:20 5:20) = 5.0
```

## axlGeo2Str

```
axlGeo2Str(  
    f_dbrep/point  
) -> t_result/nil
```

### Description

When converting floating point numbers to strings you may find the number printed is slightly differently than the value Allegro reports. This difference is due to how floating point numbers are represented in the computer. The following article is an excellent paper on the subject: *What Every Computer Scientist Should Know About Floating-Point Arithmetic* by David Goldberg.

[http://docs.sun.com/source/806-3568/ncg\\_goldberg.html](http://docs.sun.com/source/806-3568/ncg_goldberg.html)

This article also explains why sometimes the comparison of two floating numbers that appear the same results in a non-equal result. The results only differ from printf when a "5" exists at the location one place more than the database accuracy. The behavior is as follows for rounding:

- if digit at db accuracy is odd and then 5 round up
- if digit at db accuracy is even and then 5 round down

See examples below. This supports two modes, a single floating point number and a point (a list of two floating point numbers).

### Arguments

<i>dbrep</i>	a floating point number
<i>point</i>	a xy point

### Value Returned

Returns a string with Allegro rounding. If a point is passed then the return format is: "<x><xy>"

nil : not a legal argument

## See Also

[axlGeoEqual](#)

## Examples

Assume database is two decimal places

```
procedure( testit( f )
           printf("printf=%.2f\taxlGeo2Str=%s \toriginal value %.3f\n"
                  f axlGeo2Str(f) f)
           )
           testit(1.115)
           testit(1.125)
           testit(-1.115)
           testit(-1.125)

Results:
printf=1.11      axlGeo2Str=1.12          original value 1.115
printf=1.12      axlGeo2Str=1.12          original value 1.125
printf=-1.11     axlGeo2Str=-1.12        original value -1.115
printf=-1.12     axlGeo2Str=-1.12        original value -1.125
```

## Using a point

axlGeo2Str(100.124:123.345)

## **axlGeoAngleBetweenLines**

```
axlGeoAngleBetweenLines(  
    l_line1  
    l_line2  
)  
⇒ f_angle
```

### **Description**

Finds angle between two lines, in radians. If the lines are parallel or coincident, 0 is returned. The angle is the acute angle from *l\_line1* to *l\_line2*, and is positive if clockwise, negative if counter clockwise.

### **Arguments**

*l\_line1*                   List of two points defining line 1

*l\_line2*                   List of two points defining line 2

### **Value Returned**

*f\_angle*                   The angle between lines in RADIANS

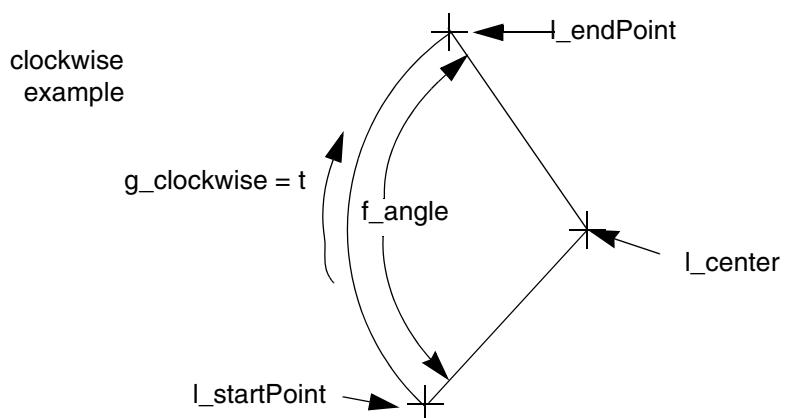
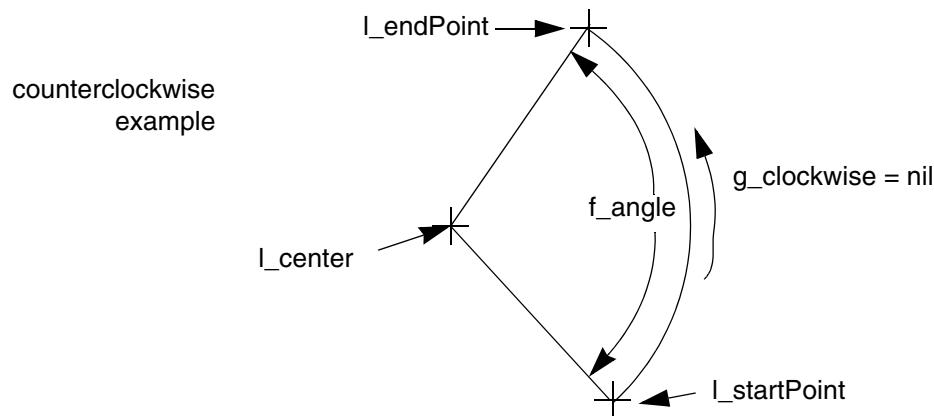
## axlGeoArcCenterAngle

```
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 25-1 on page 1482.

**Figure 25-1 Center of Arc Calculation**



## Allegro SKILL Reference

### Math Utility Functions

---

#### 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: <i>t</i> is clockwise. <i>nil</i> is counterclockwise (the default).

#### Value Returned

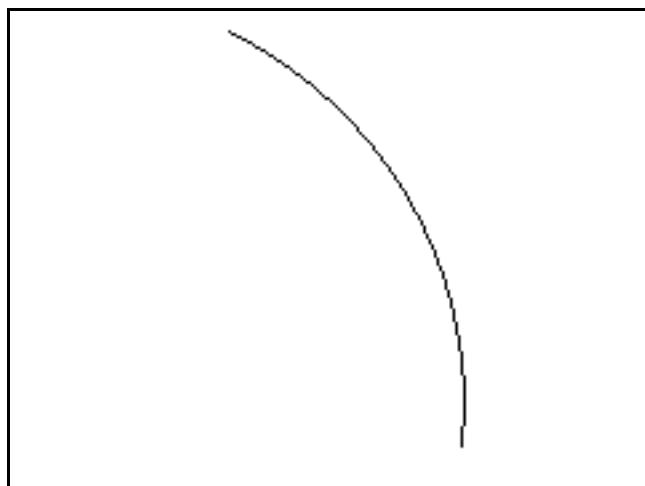
<i>l_center</i>	Center of the arc as a list: ( <i>X Y</i> ).
<i>nil</i>	Cannot calculate the center from the given arguments.

#### 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.



## **Allegro SKILL Reference**

### Math Utility Functions

---

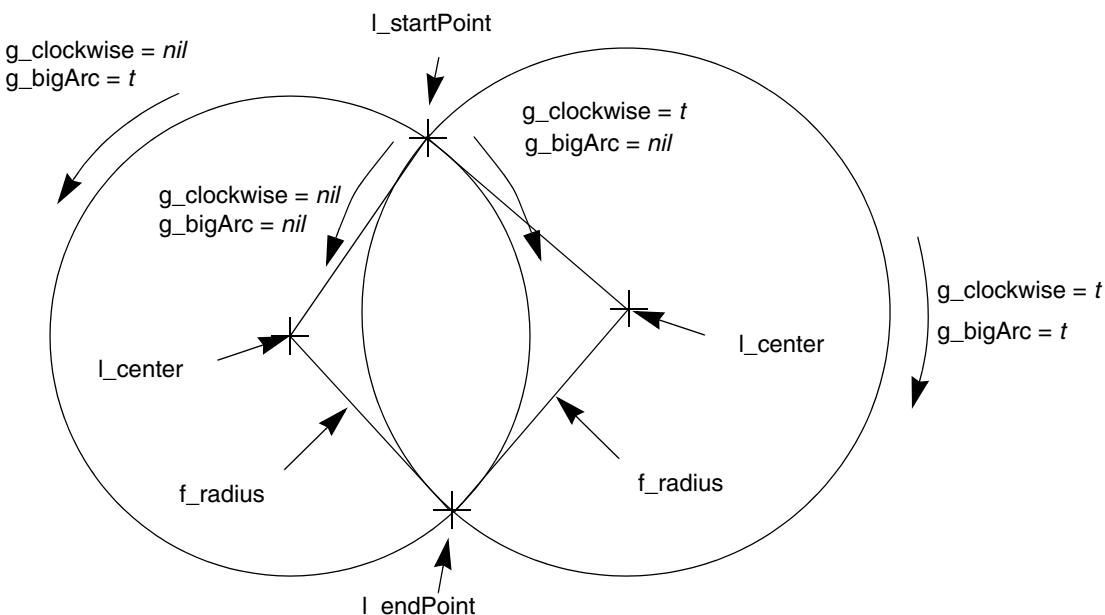
Do *Show Element* on the arc to show its center coordinate. The arc lists: "center-xy (7089, 4783)" which agrees with the (7089.086 4782.826) printed by ax1GeoArcCenterRadius.

## 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.



## Allegro SKILL Reference

### Math Utility Functions

---

#### 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
<i>g_clockwise</i>	Rotational sense of the arc: <i>t</i> is clockwise. <i>nil</i> (default) is counterclockwise.
<i>g_bigArc</i>	Flag telling whether the arc extends as the larger or the smaller of the two arcs possible between the start and endpoints.

#### Value Returned

<i>l_center</i>	Center of the arc as a list: ( <i>X Y</i> ).
<i>nil</i>	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)

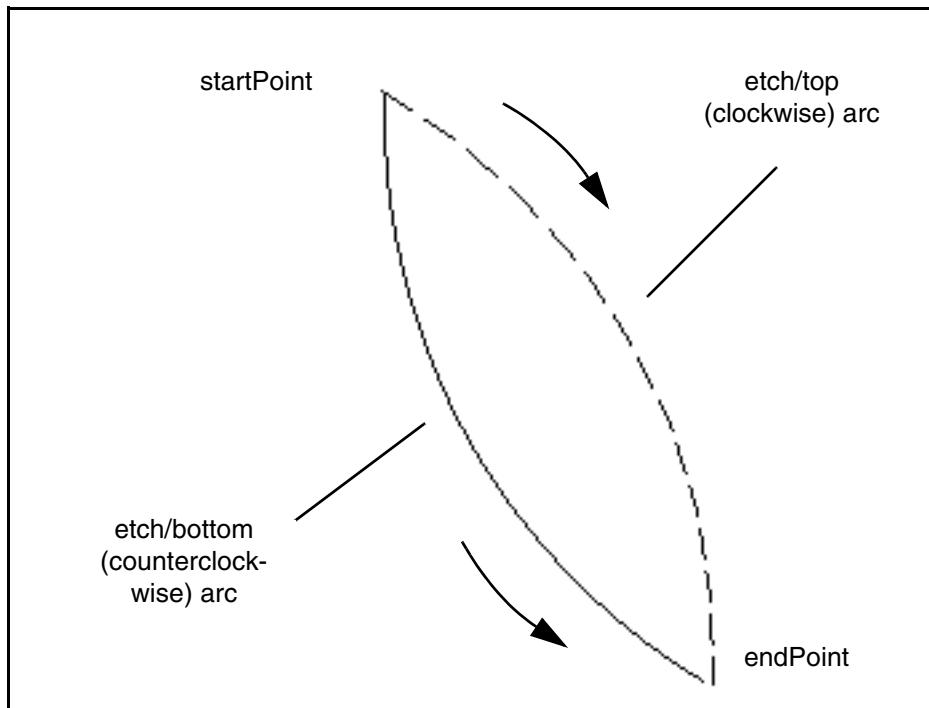
mypath = axlPathStart(list( 7500:5600))
axlPathArcRadius(mypath, 0., 8000:4700, nil, nil, 1000)
axlDBCreatePath( mypath, "etch/top")
```

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.

## Allegro SKILL Reference

### Math Utility Functions

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

```
ax1MKSConvert(  
    nil  
    [t_outUnits]  
)  
⇒ t/nil
```

Pre-registers *t\_outUnits*, the input units string, so that subsequent calls to ax1MKSConvert need not specify units.

*t\_outUnits*      String giving the units to convert to for subsequent calls to ax1MKSConvert. 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

```
ax1MKSConvert(  
    n_input  
)  
⇒ f_output/nil
```

Use this combination of arguments only after a call to ax1MKSConvert as in Arguments 3. Converts the number *n\_input* specifying a value using the *t\_outUnits* supplied by a previous call to ax1MKSConvert, and returns as *f\_output*.

*n\_input*      Number giving the input value to convert

## Allegro SKILL Reference

### Math Utility Functions

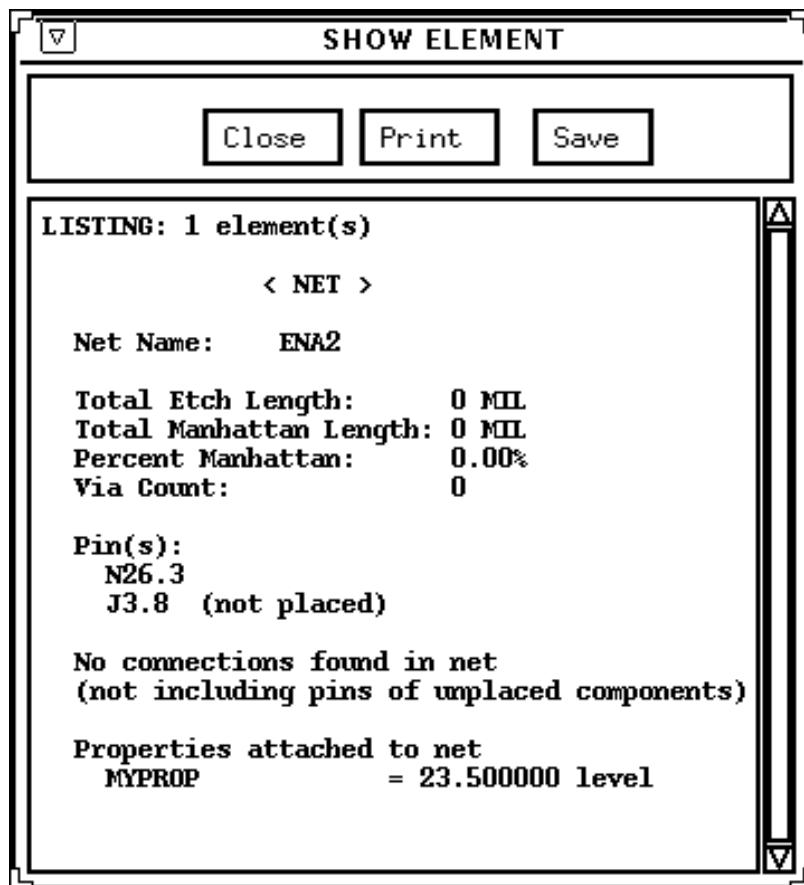
#### Value Returned 4

*f\_output*                          Converted value of *n\_input*.

#### Example 4

```
(axlMKSConvert .5) -> nil ;;error,if no preregistered units  
(axlMKSConvert nil "MILS") -> t  
(axlMKSConvert .5) -> 0.0005 ; remembers MILS
```

The following **Show Element** form shows the net with MYPROP attached:



## **axlGeoArcMidpoint**

```
axlGeoArcMidpoint(  
    l_arcPoint1  
    l_arcPoint2  
    l_arcCenter  
    f_radius  
    g_clockwize  
)  
⇒ l_midPoint/nil
```

### **Description**

Finds midpoint of an arc.

### **Arguments**

<i>l_arcPoint1</i>	Start point of arc
<i>l_arcPoint2</i>	End point of arc
<i>l_arcCenter</i>	Center of circle
<i>f_radius</i>	Radius
<i>g_clockwize</i>	Clockwise flag

### **Value Returned**

<i>l_midPoint</i>	The midpoint
<i>nil</i>	

## **axlGeoCircleCircleInt**

```
axlGeoCircleCircleInt(  
    l_point1  
    f_radius1  
    l_point2  
    f_radius2  
)  
⇒ ll_intersections/nil
```

### **Description**

Finds the intersection(s) between two circles.

### **Arguments**

<i>l_point1</i>	Center point of circle 1
<i>f_radius1</i>	Radius of circle 1
<i>l_point2</i>	Center point of circle 2
<i>f_radius2</i>	Radius of circle 2

### **Value Returned**

<i>ll_intersections</i>	A list of the intersection points
<i>nil</i>	No intersection

## **axlGeoCircleLineInt**

```
axlGeoCircleLineInt(  
    l_point1  
    l_point2  
    o_arcDbid  
)  
⇒ ll_intersections/nil
```

### **Description**

Finds the intersection(s) between a line and a circle.

### **Arguments**

<i>l_point1</i>	A point on the line
<i>l_point2</i>	Another point on the line
<i>o_arcDbid</i>	Dbid of an arc

### **Value Returned**

<i>ll_intersections</i>	A list of the intersection points
<i>nil</i>	No intersection

### **See Also**

[axlGeoCircleLineInt2](#)

## **axlGeoCircleLineInt2**

```
axlGeoCircleLineInt2(  
    l_point1  
    l_point2  
    l_center  
    f_radius  
)  
⇒ ll_intersections/nil
```

### **Description**

This function finds the intersection(s) between a line and a circle.

### **Arguments**

<i>l_point1</i>	A point on the line
<i>l_point2</i>	Another point on the line
<i>l_center</i>	Center of an arc
<i>f_radius</i>	Radius of arc

### **Value Returned**

<i>ll_intersections</i>	A list of the intersection points
<i>nil</i>	No intersection

### **See Also**

[axlGeoCircleLineInt](#)

## 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.

**Note:** To understand the basis of why a simple equal (`==`) comparison cannot always be used with floating point numbers see David Goldberg's paper on "[What Every Computer Scientist Should Know About Floating-Point Arithmetic](#)".

Wikipedia also has a good article: [http://en.wikipedia.org/wiki/Floating\\_point](http://en.wikipedia.org/wiki/Floating_point)

Floating point numbers can be internally represented by using two different formats; `float` (4 bytes) and `double` (8 bytes). All AXL interfaces use 8 bytes. SKILL interfaces depend upon the use so:

```
sscanf( "0.2" "%g" n)           ; skill float
sscanf( "0.2" "%lg" nd)         ; skill double
```

Wherever possible, use the double representation as it has more accuracy.

### 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)
```

## See Also

[axlGeo2Str](#), [axlGeoPointsEqual](#)

## **axlGeoFindAngle**

```
axlGeoFindAngle(  
    l_vector  
    l_origin  
)  
⇒ f_angle/nil
```

### **Description**

Finds the angle of a given vector.

### **Arguments**

<i>l_vector</i>	The end point from the origin
<i>l_origin</i>	The origin

### **Value Returned**

<i>f_angle</i>	The angle of the vector in RADIANS
<i>nil</i>	Vector or origin are invalid points

### **Example**

```
axlGeoFindAngle(list(1 1) list(0 0))  
=> 0.7853982
```

## **axlGeoGetBBox**

```
axlGeoGetBBox(  
    l_points  
)  
⇒ l_bbox/nil
```

### **Description**

Finds the bounding box of the given set of points.

### **Arguments**

<i>l_points</i>	The set of points
-----------------	-------------------

### **Value Returned**

<i>l_bbox</i>	The bounding box of the set of points
<i>nil</i>	Empty list

## axlGeolsBoxOverlap

```
axlGeolsBoxOverlap(  
    l_first_bbox  
    l_second_bbox  
    [t_orientation]  
)  
⇒ t/nil
```

### Description

Determines whether a box overlaps another box.

### Arguments

<i>l_first_bbox</i>	The bounding box of the first box
<i>l_second_bbox</i>	The bounding box of the second box
<i>t_orientation</i>	<ul style="list-style-type: none"><li>■ "x": check for horizontal overlap only</li><li>■ "y": check for vertical overlap only</li><li>■ nil: check for physical overlap (default)</li></ul>

### Value Returned

<i>t</i>	The two boxes overlap
nil	The two boxes do not overlap

## **axlGeoLineMidpoint**

```
axlGeoLineMidpoint(  
    l_point1  
    l_point2  
)  
⇒ l_point/nil
```

### **Description**

Finds midpoint of a line.

### **Arguments**

<i>l_point1</i>	First point
<i>l_point2</i>	Another point

### **Value Returned**

<i>l_point</i>	The midpoint
<i>nil</i>	Illegal arguments

## 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)
<i>mirror</i>	optional mirror flag.  Normally either <i>nil</i> (no mirror) or <i>t</i> for front-to-back mirror, also accepts 'GEOMETRY and 'LAYERS values, which may be set on database objects as their mirror type. You do not need to convert 'GEOMETRY to <i>t</i> and 'LAYERS to <i>nil</i> .

### Value Returned

<i>l_xyResult</i>	Rotated result.
<i>nil</i>	Error in arguments or rotation would result 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(45.0 10:200 5:2) -> (-131.4716 145.5427)
```

## **Allegro SKILL Reference**

### Math Utility Functions

---

```
axlGeoRotatePt(90.0, 100:0 nil) -> (0.0 100.0)
```

## **axlGeoPointsEqual**

```
axlGeoPointsEqual(  
    l_point1  
    l_point2  
) -> t/nil
```

### **Description**

This performs an equal comparison between two xy points and determines if they are equal within db accuracy.

### **Arguments**

two xy points

### **Value Returned**

- `t` if they are equal within current database accuracy.
- `nil` not equal

### **See Also**

[axlGeoEqual](#)

### **Examples**

```
pt1 = '(2.0 1.1)  
pt2 = '(2.0 1.1)  
axlGeoPointsEqual(pt1 pt2)
```

## **axlGeoPickShorterArc**

```
axlGeoPickShorterArc(  
    l_start  
    l_end1  
    l_end2  
    l_center  
    f_radius  
    g_clockwize  
)  
⇒ l_point/nil
```

### **Description**

Picks the shorter arc from two possibilities.

### **Arguments**

<i>l_start</i>	Start point
<i>l_end1</i>	One possible endpoint
<i>l_end2</i>	Other possible endpoint
<i>l_center</i>	Center of arc
<i>f_radius</i>	Radius of arc
<i>g_clockwize</i>	Clockwise flag of arc

### **Value Returned**

<i>l_point</i>	The point that gives the shorter arc
<i>nil</i>	Bad arguments

## **axlGeolsShorterArcClockwise**

```
axlGeolsShorterArcClockwise(  
    l_start  
    l_end  
    l_centerPoint  
)  
⇒ t/nil
```

### **Description**

Given a start point, end point and center of an arc, determines whether the clockwise or counter-clockwise arc from start to end is the shorter arc. If the arc end points are on a diameter, it returns `t` (clockwise).

### **Arguments**

<code>l_start</code>	Start point of the arc
<code>l_end</code>	End point of the arc
<code>l_centre</code>	Centre point of the arc

### **Value Returned**

<code>f_angle</code>	The angle between lines in RADIANS
----------------------	------------------------------------

## **axlMathSolveQuadratic**

```
axlMathSolveQuadratic(  
    n_a  
    n_b  
    n_c  
)  
⇒ l_roots/nil
```

### **Description**

Solves a quadratic equation in the form  $ax^2+bx+c=0$ . Does not solve equations with complex roots.

### **Arguments**

<i>n_a</i>	a in $ax^2+bx+c=0$
<i>n_b</i>	b in $ax^2+bx+c=0$
<i>n_c</i>	c in $ax^2+bx+c=0$

### **Value Returned**

<i>l_roots</i>	A list of the roots
<i>nil</i>	Complex roots

### **Example**

```
axlMathSolveQuadratic(1 2 1)  
=> (-1.0)
```

## **axlIsBetween**

```
axlIsBetween (
    g_testNum
    g_x1, g_x2
) -> t/nil
```

### **Description**

Checks if a number lies between two other given numbers. Returns *t* if a number is in a given range specified by two numbers. For example, for number *n* and range specified by *x1* and *x2*, *t* will be returned if either of  $x1 \leq n \leq x2$  or  $x1 \leq n \geq x2$  is true.

### **Arguments**

<i>g_testNum</i>	Number to test
<i>g_x1</i> , <i>g_x2</i>	Number range to test

### **Value Returned**

*t* if number is in range including the end values, otherwise *nil*

### **See Also**

[axlIsPointInsideBox](#)

## **axlIsPointInsideBox**

```
axlIsPointInsideBox(  
    l_point  
    l_box  
)  
⇒ t/nil
```

### **Description**

Checks if a point is inside a bounding box. Returns *t* if a point is inside or on the edge of a bounding box. Also see [axlGeoPointInShape](#) on page 1545 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 the 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  
    [f_nearNess]  
)  
⇒ t/nil
```

### Description

Returns *t* if point is on a given line or *nil* if not on the line.

If *f\_nearNess* value is provided, it is used as a tolerance so if a point is within the tolerance value.

In either case, an epsilon of *axlEpsilonFloat* (see *axlMathConstants*) is applied.

### Arguments

<i>l_point</i>	A point.
<i>ll_line</i>	Two end points.
[ <i>f_nearNess</i> ]	(optional) A tolerance value

### Value Returned

<i>t</i>	Point is on the specified line.
<i>nil</i>	Point is not on the specified line.

### Example

```
axlIsPointOnLine(10:20 list(5:20 15:30)) = nil  
axlIsPointOnLine(10:30 list(5:20 15:30)) = t  
axlIsPointOnLine(10:20 list(5:20 15:30) 2.5) = t
```

### See Also

[axlIsBetween](#)

## **axlLineSlope**

```
axlLineSlope(  
    ll_line  
)  
⇒ f_slope
```

### **Description**

Returns the slope of a line. You may use floating point.

### **Arguments**

*ll\_line*                  Two end points.

### **Value Returned**

*f\_slope*                  Slope of the line.

nil                        Line is vertical.

### **Example**

```
axlLineSlope(list(5.0:20.10 15.4:30.2)) = 0.9711538  
axlLineSlope(list(5:20 5:40)) = nil
```

## **axILineXLine – Obsolete**

This function is no longer required, but is kept for backward compatibility.

## axlMathConstants

Provides predefined set of high accuracy math constants. They are:

- axlPI – PI
- axlPI\_2 – PI/2.0
- axlPI\_4 – PI/4.0
- axlSQRT2 – sqrt(2)
- axlSQRT1\_2 – 1/sqrt(2)
- axlEpsilonFloat – epsilon for floating point numbers (32 bit)
- axlEpsilonDouble – epsilon for doubles (64 bit)
- axlDegPerRad – Degrees per radian
- axlRadPerDeg – Radians per degree

axlRad0, axlRad45, axlRad90, axlRad135, axlRad180, axlRad225, axlRad270, axlRad315, axlRad360 – radians values for popular degrees values

SKILL, by default, limits the number of decimal places printed but these constants are still stored and used at the full precision. To see the full precision of the constant you can do:

```
sstatus(fullPrecision t)
```

or

```
printf("%5.18f\n" axlPI)
```

## **axlMathDotProduct**

```
axlMathDotProduct (
    l_ptA1
    l_ptA2
    l_ptB1
    l_ptB2
) -> f_dotProduct
```

### **Description**

This calculates the dot or scalar product.

Dot Product:

$$(ptB1.y - ptA1.y) * (ptB2.y - ptA2.y) + (ptB1.x - ptA1.x) * (ptB2.x - ptA2.x)$$

Line are perpendicular when return is 0.0

### **Arguments**

Four coordinates describing 2 lines.

### **Value Returned**

Dot product result.

### **Example**

```
axlMidPointLine(list(5:20 15:30)) = (10.0 25.0)
```

### **See Also**

[axlDistance](#)

## **axlMidPointArc**

```
axlMidPointArc(  
    ll_endPoints  
    l_center  
    f_radius  
    g_clockwise  
) -> l_midPoint
```

### **Description**

Returns mid-point on a arc. Note mid-point may not be on a database coordinate. All parameters may be obtained from a arc dbid.

### **Arguments**

<i>ll_line</i>	arc end points
<i>l_center</i>	center of arc
<i>f_radius</i>	arc radius
<i>g_clockwise</i>	Is arc in clockwise ( <i>t</i> ) or counter clockwise direction ( <i>nil</i> )

### **Value Returned**

<i>l_midPoint</i>	mid-point of line
<i>nil</i>	error in arguments

### **See Also**

[axlMidPointLine](#)

### **Example**

- try clockwise

```
axlMidPointArc(list(20:0 0:20) 0:0 20 t) = (-14.14214 -14.14214)
```

- try counter-clockwise

```
axlMidPointArc(list(20:0 0:20) 0:0 20 nil) = (14.14214 14.14214)
```

## **axlMidPointLine**

```
axlMidPointLine(  
    ll_line  
) -> l_midPoint
```

### **Description**

Returns mid-point of line. Note mid-point might not be a database coordinate.

### **Arguments**

*ll\_line*                    Two end points

### **Value Returned**

<i>l_midPoint</i>	mid-point of line
<i>nil</i>	error in arguments

### **Example**

```
axlMidPointLine(list(5:20 15:30)) = (10.0 25.0)
```

### **See Also**

[axlMidPointArc](#)

## **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
```

## axlMUniVector

```
axlMUniVector(  
    l_pt1  
    l_pt2  
    [f_length]  
) -> l_uniPt1
```

### Description

This calculates a unit-vector. A unit vector allows one to calculate points additional points along that line.

It has two modes of operation:

- Without a length returns a unit vector to use in other operations like `axlMXYMult`. Use this mode if you need to calculate several points from the same unit vector.
- With a length, calculates a new `xy` location `f_length` from `l_pt1` along the vector specified by `pt1` and `pt2`.

This provides optimized solution over the traditional trigonometric approach.

### Arguments

`l_pt1, l_pt2`:        2 `xy` points  
`f_length`:            optional length to project

### Value Returned

Returns a unit-vector, `xy` point

`nil`: arguments were not `xy` points

### Examples

Found a point 5 units along a line from 1263.0:1062.0 to 1338.0:1137.0

```
origin = 1263.0:1062.0  
uniVec = axlMUniVector(origin 1338.0:1137.0)  
res = axlMXYMult(uniVec, 5.0 origin)
```

## **Allegro SKILL Reference**

### Math Utility Functions

---

Same as example 1 except have `uni_vec` do all the work

```
res = axlMUniVector(origin 1338.0:1137.0 5.0)
```

## **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)
```

## axIMXYMult

```
axIMXYMult(  
    l_uniVec  
    f_factor  
    [l_origin]  
) -> l_pt/nil
```

### Description

This is a convenience function that does a `l_pt.x * f_factor` and `lpt.y * factor` and returns the result. If provided an origin, it adds the origin.

```
(l_uniVec * f_factor) + l_origin
```

It is normally used in conjunction with `axIMUniVector` to project a point along a vector.

### Arguments

<code>l_uniVec</code>	xy point
<code>f_factor</code>	multiplication factor
<code>l_origin</code>	additive point

### Value Returned

<code>l_pt</code>	Returns resultant coordinate
<code>nil</code>	Arguments are not coordinates

### Examples

```
axIMXYMult(1263.0:1063.0 2.0) -> (2526.0 2126.0)
```

### See Also

[axIMUniVector](#)

## **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)
```

## **axlRadToDeg**

```
axlRadToDeg(  
    n_angle  
) => f_angle
```

### **Description**

Converts an angle in radians to degrees.

### **Arguments**

*n\_angle*                    The angle in radians

### **Value Returned**

*f\_angle*                    The angle in degrees

### **See Also**

[axlDegToRad](#), [axlMathConstants](#)

### **Example**

```
axlRadToDeg(0.7853982)  
=> 45.0
```

## **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**

<i>l_seg1</i>	1st line segment (list <i>x1:y1 x2:y2</i> )
<i>l_seg2</i>	2nd line segment (list <i>x1:y1 x2:y2</i> )

### **Value Returned**

<i>nil</i>	Error due to incorrect argument.
( <i>car l_result</i> )	Intersect or nearest point on seg1
( <i>cadr l_result</i> )	Intersect or nearest point on seg2
( <i>caddr l_result</i> )	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) 5.0)
```

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.

## bBoxAdd

```
bBoxAdd(  
    l_bBox1  
    l_bBox2  
) -> l_bBox_result
```

### Description

Adds two bounding boxes together and returns the result.

### Arguments

2 bBox values

### Value Returned

Resulting bounding box.

### Example

Expand bounding box by 100.

```
orig = '((200 100) (400 500))  
res = bBoxAdd(orig '((-100 -100) (100 100)))  
-> ((100 0) (500 600))
```

### Example 2

```
res = bBoxAdd('((200 100) (400 500)) '((0 0) (200 100)))  
-> ((200 100) (600 600))
```

---

# **Database Miscellaneous Functions**

---

## **Overview**

This chapter describes the AXL-SKILL functions that do not fit into other sections.

## axlAirGap

```
axlAirGap(  
    o_item1DBID  
    o_item2DBID/l_xy  
    [t_layer]/nil  
    [s_mode]  
)  
==> 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. Any geometric objects; logical, group or symbols are not supported (same as `show measure`). Unfilled shapes are currently treated as filled but this may change in the future.

You only need to provide a layer option when measuring between to pin or vias (also called pad comparison). When doing pad comparison without the layer, the current active layer is used. The layer syntax should either be "ETCH/<subclass>" or "<subclass>".

For spacing to the special via or pin subclasses below, either provide "PIN" or "VIA CLASS" as the class name.

- SOLDERMASK\_TOP
- SOLDERMASK\_BOTTOM
- PASTEMASK\_TOP
- PASTEMASK\_BOTTOM
- FILMMASKTOP
- FILMMASKBOTTOM

Both of these class names work equally well with pins and vias. If you want the soldermask top spacing between a pin and via, then use "PIN/SOLDERMASK\_TOP".

The second argument may be a location (in design units). Gap reports the minimum distance from the first object to this location. In enhanced mode, the location is reported as the "ETCH/TOP" layer.

Output data is returned in one of the following formats depending on the `s_mode` option:

## Allegro SKILL Reference

### Database Miscellaneous Functions

---

Default is `s_mode` (`s_mode==nil`) returns the `l_airGapData` or a `nil` if there is an error. If `s_mode` 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, Allegro 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> .

Enhanced out, `s_mode = 'enhanced` offers anyLayer air gap and returns a disembodied property list of:

`airGap` = <spacing between objects> (floating point)

`location1` = xy location first item where air gap measured

`location2` = xy location of second item where air gap measured

`layer1` = layer (class subclass) of where first object measured (string)

`layer2` = layer (class subclass) of where second object measured (string)

`isEtch` = both objects of type ETCH (boolean)

For distance between two pads, return gap based upon the active etch subclass, if `t_layer` is `nil`. Otherwise use `t_layer` to determine gap. If one or both pads do not exist on that layer:

- in anyLayer mode we will return the distance between the closest pad layers.
- it is an error in `s_mode=nil` or `s_mode=t`

For distance between a pad and non-pad element; use the layer of the pad that you want the measurement if layer is not provided we use the active layer or the top layer of the padstack.

If performance is a concern, use `anyLayer` mode over enhanced output.

The distance if objects do not share the same layer do NOT take into account board thickness.



**Tip**  
For legacy purposes, this interface does not return an air gap if the two objects do not share the same layer. If you want the air gap any layer use `s_mode = 'anyLayer` or `s_mode = 'enhanced`.

## Arguments

<i>o_item1DBID</i>	dbid of the first item.
<i>o_item2DBID</i>	dbid of the second item.
<i>l_xy</i>	The second item can be a xy location in design units and must be in the design extents.
<i>t_layer</i>	Optional layer used to resolve gap comparison between two pin or via elements.
	If in 'anyLayer or 'enhanced mode this targets a particular layer for comparison. It is most useful in measuring mask layer gaps.
<i>s_mode</i>	Return additional info to clarify error. This may be: <ul style="list-style-type: none"><li>■ nil : Default mode (objects must be on the same layer)</li><li>■ t: ("full mode") return <i>l_airGapData</i> or if not share see above (objects must be on the same layer)</li><li>■ anyLayer: support any layer measure return just gap</li><li>■ enhanced: return disembodied property list of additional air gap criteria (see above)</li></ul>

## 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 1 <sup>st</sup> item where the gap is measured.  <i>l_airGapPt2</i> : (X,Y) point on the 2 <sup>nd</sup> item where the 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(el1 el2)
-> ((1337.5 1100.0) (1362.5 1100.0) 25.0)
```

### ■ Basic input layer

```
axlAirGap(el1 el2 "TOP")
-> ((1337.5 1100.0) (1362.5 1100.0) 25.0)
```

### ■ Full output success

```
axlAirGap(el1 el2 nil t)
-> (t ((1337.5 1100.0) (1362.5 1100.0) 25.0))
```

### ■ Any layer airgap

```
q = axlAirGap(el1 el2 nil 'anyLayer)
```

### ■ Enhanced output

```
q = axlAirGap(el1 el2 nil 'enhanced)
```

### ■ Obtain soldermask spacing

```
axlAirGap(el1 el2 ""PIN/SOLDERMASK_TOP" )
-> (((1337.5 1100.0) (1362.5 1100.0) 40.0))
```

### ■ Obtain spacing to design origin

```
q = axlAirGap(el1 0:0 nil 'enhanced)
```

### ■ Full output failure

```
axlAirGap(el3 el2 nil t)
-> (RANGE ("TOP" "GND"))
```

## **axlBackDrill – Obsolete**

This interface is obsolete. Use [axlBackdrillGet](#) to retrieve actual backdrilling of pins and vias. Data returned by this interface may not match the actual backdrill results in the design.

## **axlDBGetLength**

```
axlDBGetLength(  
    o_dbid  
)  
==> f_etchlength/nil
```

### **Description**

Calculates the length of the given object which may be a NET, CLINE, SEGMENT, or RATSNEST. If RATSNEST returns the Manhattan length. If a net is partially routed, includes sum of all ratsnest Manhattan lengths.

Currently does not include VIA-Z or PIN\_DELAY in its calculation.

### **Arguments**

o\_dbid                    *dbid*

### **Value Returned**

nil                        Not a legal object

f\_etchLength             Length of object

### **See Also**

[axlDBGetManhattan](#), [axlDBPinPairLength](#)

### **Example**

```
Skill> p = ashOne()  
Skill> axlDBGetLength(p)  
-> 2676.777
```

## **ax1DBGetManhattan**

```
ax1DBGetManhattan(  
    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**

<i>l_result</i>	( <i>etchLength path manhattan</i> )
<i>nil</i>	Not a net <i>dbid</i> .
	Net is out of date.
	No ratsnest.

### **See Also**

[ax1DBGetLength](#)

### **Example**

```
p = ashOne()  
ax1DBGetManhattan(p)  
(2676.777 3300.0)
```

## **axIDBGetSymbolBodyExtent**

```
axlDBGetSymbolBodyExtent( o_dbid ) -> bBox/nil
```

## Description

This returns the body extent of a symbol. Unlike the bBox associated with a dbid, a body extent is either one of the following.

1. the extent box created by the union of all shapes on layers PACKAGE\_GEOMETRY (PLACE\_BOUND\_TOP, PLACE\_BOUND\_BOTTOM, DFA\_BOUND\_TOP, DFA\_BOUND\_BOTTOM) and EMBEDDED\_GEOMETRY (PLACE\_BOUND and DFA\_BOUND)
  2. the symbol bbox, a union of all items in symbol

The symbol instance extent box is based upon the design origin while the symdef box is based upon the symbol origin.

### Arguments

*o dbid* A symbol instance or definition

### **Value Returned**

bBox body box of symbol (minX:minY maxX:maxY)

`dbid` is not a symbol instance (symbol) or definition (symdef)

#### **See Also**

ax|DBAltOrigin

## **axlDBPinPairLength**

```
axlDBPinPairLength(  
    o_pin1  
    o_pin2  
)  
==> f_etchlength/nil
```

### **Description**

Calculate the shortest length between 2 pins. Pins must be on the same xnet. The pin can also be a VIA or RAT\_T. If the distance is not fully routed, it includes a Manhattan estimate of the unrouted portion.

Includes VIA-Z or PIN\_DELAY in its calculation if these options are enabled and if your license permits this capability.

### **Arguments**

<i>o_pin1</i>	A pin, via or rat_t
<i>o_pin2</i>	A pin, via or rat_t on same xnet as <i>o_pin1</i>

### **Value Returned**

*nil* – Not a legal object; unsupported dbid or items not on same xnet

*f\_etchLength* – length of object

### **See Also**

[axlDBGetLength](#)

### **Example**

```
Skill> pin1 = ashOne()  
Skill> pin2 = ashOne()  
Skill> axlDBPinPairLength(pin1 pin2)  
-> 2676.777
```

## **axlDeleteByLayer**

```
axlDeleteByLayer(  
    t_layerName/lt_layerName  
    [nil/'fixed']  
)  
==> x_cnt/nil
```

### **Description**

Deletes all data on one or more provided layers. The following should be noted:

- Does not delete pins or vias.
- Deletes pins escapes and other symbol data associated with symbols.
- Does not delete objects on a symbol definition. If you are using this interface as a prerequisite to deleting a layer, objects on a symbol definition may prevent you from deleting the layer.
- To delete dynamic shapes, you also need to delete data on the equivalent BOUNDARY class.
- Certain classes, such as, DRC\_ERROR\_CLASS, PIN, VIA\_CLASS, ROUTER\_PLAN and CAVITY, are ignored.

### **Arguments**

<i>t_layerName</i>	layer name <class>/<subclass>
<i>lt_layerName</i>	list of layer names
<i>'fixed</i>	Optional, ignore FIXED property

### **Value Returned**

<i>x_cnt</i>	number of items deleted
<i>nil</i>	any error

### **Example**

- Delete all data on ETCH/TOP except for fixed data

```
axlDeleteByLayer("ETCH/TOP")
```

- Delete all data on ETCH/BOTTOM plus OUTLINE layers including fixed

```
axlDeleteByLayer(list("ETCH/TOP" "BOARD GEOMETRY/OUTLINE") 'fixed)
```

## axlExtentDB

```
axlExtentDB()
⇒ l_bBox/nil
```

### Description

Determines a design type and returns the *bBox* extent.

For a layout this function 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.

For a symbol drawing (a .dra file), this function computes the bounding box enclosing all objects visible for a drawing.

### Arguments

None

### Valued Returned

<i>l_bBox</i>	Returns <i>bBox</i> extent.
nil	Unknown drawing type.

## **axlExtentLayout – Obsolete**

This function is obsolete. Use `axlExtentDB`. Kept for backward compatibility.

## **axIExtentSymbol – Obsolete**

This function is obsolete. Use [axIExtentDB](#). Kept for backward compatibility.

## axlFindPath

```
axlFindPath(  
    o_oneDbid  
    o_twoDbid  
    [g_altPath]  
)  
==> lo_dbid/llo_dbid/nil
```

### Description

Finds an etch path from one object to another. Items must be on the same net and must be connect type, such as, pins, vias, clines or shapes, and tee.

#### Restrictions:

- A partial connection between the 2 objects (ratsnest still exists) results in a nil return.
- Segments are promoted to their owning cline (path)

Return list is ordered by:

```
o_oneDbid, ... <connected items>, o_twoDbid
```

To use this for finding loops on a net, you must compare every node to every other node. This can be very time consuming for large pin count nets.



***– If multiple paths exist between the two objects, returns will follow a single path but the one it uses is not defined (by this it may decide on the shortest or longest).***

***– Because of the high number of interconnects, VOLTAGE nets may not return correct results since the algorithm is recursive and terminates if it nests too deeply.***

## Arguments

<i>o_oneDbid</i>	first net item
<i>o_twoDbid</i>	second net item
[ <i>g_altPath</i> ]	enable alternate path

## Value Returned

<i>nil</i>	no path exists between objects or an error
<i>lo_dbid</i>	path list if <i>g_altPath</i> is nil
<i>llo_dbid</i>	path list if <i>g_altPath</i> is t. First item is one path and second item is nil or the alternative path. ( <i>lo_1dbid l01dbid</i> )

## Example

### 1. Find a path between two items

```
; ashOne is a selection utility found at <cdsroot>/pcb/examples/skill/ash-fxf/  
ashone.il  
one = ashOne()  
two = ashOne()  
; pick a line, cline or segment (set find filter)  
path = axlFindPath(one two)  
axlShowObject(path)
```

### 2. See if the two objects is a start/end point of a loop

```
path = axlFindPath(one two t)
```

## **axlGeoClosestPointOnArc**

```
axlGeoClosestPointOnArc(  
    l_point  
    o_arcDbid  
)  
⇒ l_closest_point/nil
```

### **Description**

Finds the closest point from the given arc to the given point.

### **Arguments**

<i>l_point</i>	A point
<i>o_arcDbid</i>	Dbid of an arc

### **Value Returned**

<i>l_closest_point</i>	The closest point on the arc to the given point
<i>nil</i>	Invalid parameters

## **axlGeoClosestPointOnCircle**

```
axlGeoClosestPointOnCircle(  
    l_point  
    l_center  
    l_radius  
)  
⇒ l_closest_point/nil
```

### **Description**

Finds the closest point from the given circle to the given point.

### **Arguments**

<i>l_point</i>	A point
<i>l_center</i>	Center point of circle
<i>l_radius</i>	Radius of circle

### **Value Returned**

<i>l_closest_point</i>	The closest point on the arc to the given point
<i>nil</i>	Invalid parameters

### **See Also**

[axlGeoClosestPointOnArc](#)

## axlGeoPointInShape

```
axlGeoPointInShape(  
    l_point  
    o_dbid/o_polygon  
    [g_include_voids]  
    [t/nil]  
)  
⇒ t/nil
```

### Description

Given a point and a shape *dbid*, determines whether that point is inside or outside the shape or a polygon. For a shape with voids, a point is considered *outside* the given shape if inside a void. If shape has voids and *g\_include\_voids* is *t* then point is outside if inside a void.

The command does not allow hole polygons as input. When polygon holes is passed the following warning is displayed:

```
Invalid polygon id argument -<argument>
```

### Arguments

<i>l_point</i>	Point to check.
<i>o_dbid/o_polygon</i>	dbid of the shape / <i>o_polygon</i>
[ <i>g_include_voids</i> ]	Applicable only in case the second parameter is a shape otherwise it's ignored.  In case of shapes, if the parameter value is nil, voids are excluded. The default value is <i>t</i> .
[ <i>t/nil</i> ]	<i>t</i> means include voids, <i>nil</i> means use the shape outline only.
Default is <i>t</i> .	

### Value Returned

<i>t</i>	Point is inside the shape.
<i>nil</i>	Point is outside the shape, or incorrect arguments were given.

### See Also: [axlGeoPointShapeInfo](#)

## axlGeoPointShapeInfo

```
axlGeoPointShapeInfo(  
    l_point  
    o_dbid  
) ==> (g_state o_dbid)/nil
```

### Description

Given a point and a shape dbid returns relation of point to shape. State may be outside, inside or on. Additional dbid is returned in the second argument to indicate if void or shape is involved.

Return matrix:

G_STATE	O_DBID
outside	nil if outside shape, void dbid if inside void
inside	nil
on	shape dbid if on shape else void dbid



- Assumes that cross-hatch shapes are solid filled.
- Rounds point to database units. If database accuracy is 2 and you pass a 3 decimal place point, we will round it to 2 places before doing the test.

### Arguments

l_point	the point
o_dbid	dbid of the shape

### Value Returned

nil - if an error since as an invalid argument

g\_state/o\_dbid - see *Description*

## **axlGetImpedance**

```
axlGetImpedance(  
    o_dbid  
) => (f_min f_max)/nil
```

### **Description**

Returns minimum and maximum impedance for given item. Item can be either cline, cline segment, net or xnet. Impedance is in ohms by default.

### **Arguments**

<i>o_dbid</i>	Segment cline
---------------	---------------

### **Value Returned**

<i>f_min f_max</i>	Impedance in current MKS units.
<i>nil</i>	Segment is not a cline segment.

### **See Also**

[axlSegDelayAndZ0](#)

## **axlGeoMirrorLayer**

```
axlGeoMirrorLayer(  
    t_layer  
    t_mirror  
)  
⇒ t_result/nil
```

### **Description**

Mirrors a subclass to get the target layer. Given a layer and one of the mirror codes, the function returns the layer, which maps based on the mirroring.

### **Arguments**

<i>t_layer</i>	Name of the starting layer
<i>t_mirror</i>	Mirror code to apply ('GEOMETRY, 'LAYERS, t, nil).

### **Value Returned**

<i>t_result</i>	Layer to put objects on after mirroring.
nil	Error in arguments or layer doesn't exist

### **Example**

```
axlGeoMirrorLayer("ETCH/TOP" t) -> "ETCH/BOTTOM";
```

## **axlImpdedanceGetLayerBroadsideDPImp**

```
axlImpdedanceGetLayerBroadsideDPImp (
    t_layer1/x_layerNum1
    t_layer2/x_layerNum2
    f_width
) ==> f_diffImpedance/nil
```

### **Description**

Computes the differential impedance of a broadside-coupled diffpair with the given line width and two specified layers on which the signal lines will be routed. A warning message may be given if the parameters are inappropriate for the calculation.

### **Arguments**

t_layer1	Layer name (example "ETCH/TOP" or "TOP")
x_layerNum1	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
t_layer2	Layer name (example "ETCH/TOP" or "TOP")
x_layerNum2	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
f_width	The line width in user units.

### **Value Returned**

The line differential impedance in ohms (float) or nil on error.

### **See Also**

[axlImpedance2Width](#)

## **axlImpdedanceGetLayerBroadsideDPWidth**

```
axlImpdedanceGetLayerBroadsideDPWidth(
    t_layer1/x_layerNum1
    t_layer2/x_layerNum2
    f_diffImpedance
)
==> f_lineWidth/nil
```

### **Description**

Computes the differential impedance of a broadside-coupled diffpair with the given line width and two specified layers on which the signal lines will be routed. A warning message may be given if the parameters are inappropriate for the calculation.

### **Arguments**

t_layer1	Layer name (example "ETCH/TOP" or "TOP")
x_layerNum1	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
t_layer2	Layer name (example "ETCH/TOP" or "TOP")
x_layerNum2	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
diffImp	The target differential impedance in ohms.

### **Values Returned**

The line width in user units or nil on error.

### **See Also**

[axlImpedance2Width](#)

## **axlImpdedanceGetLayerEdgeDPImp**

```
axlImpdedanceGetLayerEdgeDPImp (
    t_layer/x_layerNum
    f_spacing
    f_width
) ==> f_diffImpedance/nil
```

### **Description**

Computes the differential impedance of a edge-coupled diffpair with the given line width and spacing on a specified layer. A warning message may be given if the parameters are inappropriate for the calculation.

### **Arguments**

t_layer	Layer name (example "ETCH/TOP" or "TOP").
x_layerNum	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
f_spacing	Spacing between the two signal lines in use units.
f_width	The line width in user units.

### **Value Returned**

The differential impedance value in ohms (float) or `nil` on error.

### **See Also**

[axlImpedance2Width](#)

## **axlImpdedanceGetLayerEdgeDPSpacing**

```
axlImpdedanceGetLayerEdgeDPSpacing (
    t_layer/x_layerNum
    f_width
    f_diffImp
)
==> f_spacing/nil
```

### **Description**

Given the line width of the two signal lines of an edge-coupled diffpair on the specified layer, finds the spacing such that the differential impedance is closest to the target value. A warning message may be given if the parameters are inappropriate for the calculation.

### **Arguments**

t_layer	Layer name (example "ETCH/TOP" or "TOP").
x_layerNum	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
f_width	The given line width, in user units.
f_diffImp	The target differential impedance in ohms.

### **Value Returned**

The target spacing in user units or `nil` on error.

### **See Also**

[axlImpedance2Width](#)

## **axlImpdedanceGetLayerEdgeDPWidth**

```
axlImpdedanceGetLayerEdgeDPWidth (
    t_layer/x_layerNum
    f_spacing
    f_diffImp
) ==> f_width/nil
```

### **Description**

Given the spacing of the two signal lines of an edge-coupled diffpair on the specified layer, finds the line width such that the differential impedance is closest to the target value. A warning message may be given if the parameters are inappropriate for the calculation.

### **Arguments**

t_layer	Layer name (example "ETCH/TOP" or "TOP").
x_layerNum	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
f_spacing	The spacing between the two signal lines in user units.
f_diffImp	The target differential impedance in ohms.

### **Value Returned**

The line width in database units (float) or `nil` on error.

### **See Also**

[axlImpedance2Width](#)

## **axlImpedance2Width**

```
axlImpedance2Width(  
    t_layer/x_layerNum  
    f_impedance  
) ==> f_lineWidth/nil
```

### **Description**

Converts the given impedance on a specified layer to a line width.

**Note:** None of the axlImpedance APIs are available in Allegro PCB L.

### **Arguments**

t_layer	Layer name (example "ETCH/TOP" or "TOP").
x_layerNum	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
f_impedance	The impedance value, in ohms, that is to be converted to a line width.

### **Value Returned**

f_lineWidth	The converted line width in drawing units.
nil	Conversion was not successful.

### **See Also**

[axlImpedance2Width](#)  
[axlImpedanceGetLayerEdgeDPImp](#)  
[axlImpedanceGetLayerEdgeDPWidth](#)  
[axlImpedanceGetLayerEdgeDPSpacing](#)  
[axlImpedanceGetLayerBroadsideDPImp](#)  
[axlImpedanceGetLayerBroadsideDPWidth](#)

## axlPadOnLayer

```
axlPadOnLayer(  
    o_dbid  
    t_layer/x_layerNumber  
    [g_noPadSuppress]  
)  
==> t/nil
```

### Description

Tests if a pad is present on an etch layer. A pad is present on the layer if the padstack has a regular, anti or thermal pad and it is not suppressed by the rules of Pad Suppression.

While this does support a padstack dbid, for best operation, pass the VIA or PIN object.

### Arguments

<i>o_dbid</i>	A via, pin or padstack
<i>t_layer</i>	Name of layer (for example, "TOP")
<i>x_layerNumber</i>	layer number (starts at 0)
<i>g_noPadSuppress</i>	t if ignore pad suppression, nil (default) use pad suppression

### Value Returned

*t* if a pad is on layer; *nil* no pad on layer

### See Also

[axlPadSuppressGet](#)

### Example

- Using ashOne shareware in <cdsroot>/share pcb/examples/skill/ash-fxf/ashone.il

Assuming a design where pad suppression is enabled on etch layer GND

```
pad = ashOne(list("vias" "pins"))
```

```
res1 = axlPadOnLayer(pad "GND")
```

## **Allegro SKILL Reference**

### Database Miscellaneous Functions

---

```
res2 = axlPadOnLayer(pad "GND" t)
```

## **axlPinExport**

```
axlPinExport(  
    g_includeTextLocation  
    [t_csvfile]  
)  
--> t/nil
```

### **Description**

This exports all pins in the symbol editor in csv format. The format of the csv file is described in [axlPinImport](#).

**Note:** Function is only enabled in symbol editor.

### **Arguments**

<i>g_includeTextLocation</i>	if <i>t</i> , include pin text location (offset, rotation and mirror); <i>nil</i> omits data which means pin text when loaded into a symbol will be located at pin origin.
<i>t_csvfile</i>	Name of csv file; default is symbol name. Assumes a csv extension.

### **Value Returned**

<i>t</i>	csv file created
<i>nil</i>	failed to create csv file

### **See Also**

[axlPinImport](#)

### **Example**

See example in [axlPinImport](#).

## **axlPinImport**

```
axlPinImport(  
    t_csvFile  
)  
--> l_cnt/nil
```

### **Description**

This imports pin csv (comma separated values) file into the symbol editor. With this file you can describe the location and other characteristics of a set of pins (including mechanical) that comprise a symbol.

**Note:** This function is only enabled in symbol editor.

To best understand the format of this file, you should export one via axlPinExport.

Two formats are supported:

- pin only, pin text is located at pin origin
- pin with text

File format:

- A '#' indicates a comment
- (Optional) Units,<*units strings*>
- table describing pins

Pin Table (column number indicated)

1. PinNumber - Pin number, if blank then a mechanical pin.
2. Padstack - name of padstack
3. x - x location of pin (no units)
4. y - y location of pin (no units)
5. rotation - pin rotations, if blank has no rotations

If the pin text option is used then the following columns should be present.

1. x offset location from pin origin
2. y offset location from pin origin

3. rotation of text (absolute), if blank no rotation
4. textMirror; blank no mirror, "m" text should be mirrored

Text block used for pin text is the design active text block.

**Note:** Setting axlDebug() may give additional info on why pins fail to load.

## Arguments

*t\_csvFile* csv file, assumes a .csv extension.

## Value Returned

<i>nil</i>	Unable to open file or no pins loaded
<i>l_cnt</i>	A list of ( <i>x_pinsLoaded</i> <i>x_pinFailed</i> )

## See Also

[axlPinExport](#)

## Example

- In the symbol editor with a *.dra* file loaded. Export pins with text location, date, delete all pins and then import them:

```
axlPinExport(nil "foo")
axlDeleteObject(axlDBGetDesign()->pins nil
axlPinImport("foo")
```

## **axlReratNet**

```
axlReratNet(  
    t_netName/o_dbid  
)  
==> t/nil
```

### **Description**

Rerats a net. Normally this is not required since Allegro PCB Editor automatically updates ratsnesting as required.

### **Arguments**

<i>t_old_name</i>	the existing net name.
<i>o_dbid</i>	Alternative is a dbid that is on a net

### **Value Returned**

<i>t</i>	the net is successfully reratted.
<i>nil</i>	fails.

### **Example**

```
axlReratNet("NET1")
```

## axlText2Lines

```
axlText2Lines(  
    o_textDbid  
)  
==> llr_path/nil
```

### Description

This vectorizes a text dbid into a list of lists of `r_path` objects.

The return is a list of list `r_paths` for each character:

```
llr_path = (l_rpathChar1, l_rpathChar2 ... l_lrpath_CharLast)
```

Each character can have one or more line draws and each line draw can have one or more segments. For example, an 'A' has 2 line draws; one have 2 segments and the second 1 segment.

```
l_rpathChar1 = (l_rpathLine1, ... lrpathLineN)
```

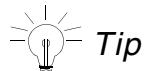
where:

```
l_rpathLineX->_width -> thickness of line  
l_rpathLineX->_pathList -> list of segments making up a line
```

Things to note:

- Vectorization returns line segments (no arcs) although this may change in the future.
- A single character may return multiple `r_paths` and one `r_path` may have multiple segments.
- The width is the same for all lines making up a single `textDbid`. This means that the width for all segments undefined since the `r_path` has the width.
- Characters are returned left to right.
- Whitespace is skipped.

Allegro draws all text as stroke text. This converts a text dbid into a series of line draws using `r_path` structures.



*Tip*  
You can convert a `r_path` to an `o_polygon` by using `axlPolyFromDB` using its "`?line2poly t`" option.

## Arguments

*o\_textDbid*                    A text dbid

## Value Returned

<i>llr_path</i>	A list of list of <i>r_paths</i> (see above)
<i>nil</i>	An error (not a text dbid) or text dbid is an empty string (shown in Allegro with a small triangle).

## See Also

[axlPolyFromDB, Path Functions](#)

## Example

Function *ashOne* is a shareware utility that allows user to select one object (see <cdsroot>/share/pcb/examples/skill/ash-fxf/ashone.il).

■ Pick a text and add converted lines on BOARD GEOMETRY/OUTLINE layer

```
text = ashOne("TEXT")
lines = axlText2Lines(text)
layer = "BOARD GEOMETRY/OUTLINE"
; flatten list
flattened = foreach( mapcan x lines x)
; create objects in database
foreach(path flattened i = axlDBCreatePath(path layer nil nil nil))
```

■ Pick a text and add converted to shapes on "BOARD GEOMETRY/ASSEMBLY\_DETAIL"

```
text = ashOne("TEXT")
lines = axlText2Lines(text)
layer = "BOARD GEOMETRY/ASSEMBLY_DETAIL"
; flatten list
flattened = foreach( mapcan x lines x)
foreach(path flattened
; may return multiple polys
polys = axlPolyFromDB(path ?endCapType 'ROUND ?line2poly t)
; create shapes in database
foreach(poly polys i = axlDBCreateShape(poly t layer nil nil)))
)
```

## **axlUnfixAll**

```
axlUnfixAll(  
    )  
==> x_count
```

### **Description**

This is a convenience API.

If removes the FIXED property from all elements in the design.

### **Arguments**

none

### **Value Returned**

<i>x_count</i>	Number of fixed properties removed.
----------------	-------------------------------------

### **Example**

```
axlUnfixAll()
```

## **axlWidth2Impedance**

```
axlWidth2Impedance(  
    t_layer/x_layerNum  
    f_lineWidth  
) ==> f_impedance/nil
```

### **Description**

Converts the given line width on a specified layer to an impedance. This uses the field solver to compute the impedance

### **Arguments**

t_layer	Layer name (example "ETCH/TOP" or "TOP").
x_layerNum	Number of the etch subclass. Layers are numbered starting with 0 for the Top layer.
f_lineWidth	The line width to be converted to an impedance.

### **Value Returned**

f_impedance	The converted impedance value.
nil	Conversion was not successful.

### **See Also**

[axlImpedance2Width](#)

## **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 Allegro 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.

Uses current testprep parameter settings except (these may be relaxed in future releases):

- set to flood
- set allow SMT/Blind or Thru pad stack type

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. This may be a future enhancement. For the present, use axlDeleteObject and axlDBGetAttachedText.

Supports axlDebug API to print failure to place error.

## Arguments

*o\_dbid* Pin or via *dbid*  
*g\_mode* Add test point to top or bottom, or clear one.

## Value Returned

*t* Object changed.  
*nil* Error other than test point checks.  
*s\_error* 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_ARE A	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

## Examples

The following examples use the `ashone.il` file in `<cdsroot>/share pcb/skill/examples` to allow you to select objects:

### 1) Add testpoint to top

```
axlUIWPrint(nil 'infol "Select pin or via to add testpoint")  
dbid = ashOne('VIAS PINS))  
ret = axlTestPoint(dbid 'top)
```

### 2) Clear a testpoint

```
axlUIWPrint(nil 'infol "Select pin or via to clear testpoint")  
dbid = ashOne('VIAS PINS))  
ret = axlTestPoint(dbid nil)
```

## **axlChangeNet**

```
axlChangeNet (
  o_dbid
  t_netName/o_netdbid
)
⇒ t/nil
```

### **Description**

Changes the net an object is currently on. Restricted to shapes, filled rectangles (rectangles), pins and vias. Returns t when successful. Will not rip up clines or vias.

Failure can occur for the following reasons:

- Object is not supported.
- netName does not exist.

The following restrictions apply to this function:

- Pins must be assigned. Pins must have an associated component. Mechanical pins are un-assigned.
- Via net assignment is advised. The via must be able to connect to something on the provided net to remain on that net. Otherwise, it will fall back to the original net or possibly another net.
- If a via is in open space, it will be on a dummy net. This API cannot be used to force it onto a net.
- This API is useful for a via, if it touches multiple shapes but it is assigned to the wrong shape's net.

Potential side effects of this function:

- It may not properly reconnect two touching cline segments that were previously connected by the shape.
- Clines only attached to the shape will inherit the new net of the shape.
- Vias attached to the shape will not inherit the new net. This is different from the Allegro change net command.

## **Allegro SKILL Reference**

### Database Miscellaneous Functions

---

#### **Arguments**

<i>o_dbid</i>	Shape <i>dbid</i>
<i>t_netName/</i> <i>o_netcdbid</i>	Name of a net or a netdbid (for dummy nets)

#### **Value Returned**

<i>t</i>	Object changed.
<i>nil</i>	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**

<i>f_delay</i> <i>f_z0</i>	Delay and impedance in current MKS units.
<code>nil</code>	Segment is not a cline segment.

### **See Also**

[axlGetImpedance](#)

## **axlSetDefaultDieInformation**

```
axlSetDefaultDieInformation(comp)
  ==> t/nil
```

### **Description**

Sets the default die information for a component.

This function will configure a newly-placed IC-class component as a die in an MCM design. Based on the placed component's information the die will be flagged as either wire bond or flip-chip.

### **Arguments**

comp	dbid of the component / symbol to set default information for.
------	--

### **Value Returned**

t if successful, nil otherwise.

---

# **Microsoft Excel Integration Functions**

---

## **axlSpreadsheetClose**

```
axlSpreadsheetClose()  
  ==> t
```

### **Description**

Releases the spreadsheet document in memory. All information is freed. This function should be called whenever you have completed working with the active spreadsheet document.

Once the spreadsheet information is released, you cannot access any data about it. This includes retrieving style information, cell contents, lists of worksheets, etc. Any such information that you need to reference after the spreadsheet is freed should be retrieved prior to this call.

If there is no active spreadsheet, this function does nothing.

### **Arguments**

Nothing

### **Values Returned**

t	Spreadsheet information successfully freed.
---	---

### **Example**

The following example creates a simple spreadsheet, adds information to the first cell ("Hello"), writes the spreadsheet, and closes it.

```
axlSpreadsheetInit()  
  ==> t
```

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

```
a xlSpreadsheetSetWorksheet("First")
  ==> t

a xlSpreadsheetDefineCell(1 1 "Default" "String" "Hello")
  ==> t

a xlSpreadsheetWrite("example.xml")
  ==> t

a xlSpreadsheetClose()
  ==> t
```

#### **See Also**

[a xlSpreadsheetInit](#), [a xlSpreadsheetRead](#), [a xlSpreadsheetWrite](#)

## **axlSpreadsheetDefineCell**

```
axlSpreadsheetDefineCell(  
    x_row  
    x_col  
    t_style  
    t_type  
    t_value)  
==> t / nil
```

### **Description**

Completely defines a single cell in the active worksheet. This function is more efficient than calling [axlSpreadsheetSetCell](#) with multiple [axlSpreadsheetSetCellProp](#) calls afterwards.

### **Arguments**

x_row	Row index (1-based) for the desired cell.
x_col	Column index (1-based) for the desired cell.
t_style	Style name to apply to this cell / nil for default.
t_type	Type definition for this cell / nil for default (string).
t_value	Value for cell / nil for empty.

### **Value Returned**

t	Cell successfully defined.
nil	Cell not defined. See console for reason.

### **Example**

The following example sets the contents of cell 1, 1 in the active worksheet to be the string "Hello" using the default style.

```
axlSpreadsheetDefineCell(1 1 "Default" "String" "Hello")  
==> t
```

### **See Also**

[axlSpreadsheetGetCell](#), [axlSpreadsheetSetCell](#), [axlSpreadsheetSetCellProp](#)

## **axlSpreadsheetDoc**

### **Description**

The axlSpreadsheet family of functions allow you to read and write Microsoft's open XML-based spreadsheet format from within SKILL. You can create a spreadsheet from data within your active Allegro tool, or you can read a spreadsheet and extract information from it to update your database.

Documentation for individual functions is separately available. This entry provides an overview, as well as a small example of how to use the API routines together.

### **Example**

The following is a simple example which creates a small, two-worksheet spreadsheet with a few formatting style definitions and cells which use those styles to format their contents when the spreadsheet is viewed with a tool such as Microsoft's Excel.

```
procedure( spreadsheetExample() ; Initialize an empty spreadsheet.  
          ; Note that you do not need to provide a name until you  
          ; wish to write the spreadsheet to disk.  
          axlSpreadsheetInit()  
  
          ; Define initial, default style.  
          ; Styles may be defined at any point during the spreadsheet's  
          ; construction, but must be defined before they are referenced  
          ; by any row, column, or cell.  
          axlSpreadsheetSetStyle("Default" nil)  
          axlSpreadsheetSetStyleProp("Alignment" "Vertical" "Top")  
          axlSpreadsheetSetStyleProp("Alignment" "Horizontal" "Left")  
          axlSpreadsheetSetStyleProp("Alignment" "WrapText" "1")  
  
          ; Define a second style, derived from the Default style, which  
          ; will include a thin border outline and specifies a red  
          ; background fill.
```

## Allegro SKILL Reference

### Microsoft Excel Integration Functions

---

```
axlSpreadsheetSetStyle("Red" "Red Cell")
axlSpreadsheetSetStyleParent("Default")
axlSpreadsheetSetStyleBorder("Left" nil "Continuous" "2")
axlSpreadsheetSetStyleBorder("Right" nil "Continuous" "2")
axlSpreadsheetSetStyleBorder("Top" nil "Continuous" "2")
axlSpreadsheetSetStyleBorder("Bottom" nil "Continuous" "2")
axlSpreadsheetSetStyleProp("Fill" "Color"
axlSpreadsheetGetRGBColorString(255 0 0))
axlSpreadsheetSetStyleProp("Fill" "Pattern" "Solid")

; Define the first worksheet in the spreadsheet.
axlSpreadsheetSetWorksheet("First")
; With a wider first column
axlSpreadsheetSetColumnProp(1 "Width" "500")
axlSpreadsheetDefineCell(1 1 "Default" "String" "Default formatted cell")
axlSpreadsheetDefineCell(1 2 "Red" "String" "Red background cell")

; Write the compiled spreadsheet to XML file on disk.
axlSpreadsheetWrite("spreadsheet.xml")

; Close and release the compiled spreadsheet's data.
axlSpreadsheetClose()
)
```

## **axlSpreadsheetGetCell**

```
axlSpreadsheetGetCell(  
    x_row  
    x_col  
) ==> g_cellData/ nil
```

### **Description**

Retrieves the data from the specified cell.

### **Arguments**

x_row	Row index (1-based) of cell to look up.
x_col	Column index (1-based) of cell to look up.

### **Value Returned**

g_cellData	Structure defining the contents cell
nil	Cell contents are currently empty or undefined.

### **Example**

The following example reads a spreadsheet into memory, then gets the contents of one cell from the first worksheet.

```
axlSpreadsheetRead("example.xml")  
    ==> t  
  
axlSpreadsheetSetWorksheet("First")  
    ==> t  
  
g_cell = axlSpreadsheetGetCell(1 1)  
  
g_cell->??  
    ==> (column 4  
        row    1  
        data   "a"  
        type   "String"  
        style  "Default"  
    )  
  
axlSpreadsheetClose()  
    ==> t
```

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

#### **See Also**

[axISpreadsheetSetCell](#), [axISpreadsheetSetCellProp](#), [axISpreadsheetDefineCell](#)

## **axISpreadsheetGetRGBColorString**

```
axISpreadsheetGetRGBColorString(  
    x_red  
    x_green  
    x_blue)  
==> t_rgb / nil
```

### **Description**

Given red, green, and blue color values, return an RGB string for use in spreadsheet style definitions in format required for Microsoft open spreadsheet format.

### **Arguments**

x_red	Integer red value (0-255)
x_green	Integer green value (0-255)
x_blue	Integer blue value (0-255)

### **Value Returned**

t_rgb	String denoting the color value for the RGB value passed in.
nil	Illegal values passed (outside legal range).

### **Example**

The following example get the RGB string value associated with pure red.

```
axISpreadsheetGetRGBColorString(255 0 0)  
==> "#ff0000"
```

### **See Also**

[axISpreadsheetGetRGBColorString](#)

## **axlSpreadsheetGetRGBForNamedColor**

```
axlSpreadsheetGetRGBForNamedColor(  
    t_name  
)  
==> t_rgb / nil
```

### **Description**

Spreadsheets have a small set of known, pre-defined color values. To retrieve the RGB value for a specific named color, pass that color name to this function.

### **Arguments**

t\_name                    Name of color to retrieve RGB value for.

### **Value Returned**

t\_rgb                    String denoting RGB value for the color value passed in.  
nil                      Color name was not found in list of standard colors.

### **Example**

The following example get the RGB string value associated with the predefined color name "cyan".

```
axlSpreadsheetGetRGBForNamedColor("cyan")  
==> "#00FFFF"
```

### **See Also**

[axlSpreadsheetGetRGBColorString](#)

## **axlSpreadsheetGetStyles**

```
axlSpreadsheetGetStyles()  
  ==> l_styles / nil
```

### **Description**

Retrieves a list of all the styles defined for the active spreadsheet. If no worksheets currently exist, nil is returned.

### **Arguments**

none

### **Value Returned**

l_styles	List of style names and IDs as pairs (ID, name)).
nil	No worksheets current defined / no spreadsheet active.

## **EXAMPLES**

The following example reads a spreadsheet into memory, then gets the list of defined styles for that spreadsheet.

```
axlSpreadsheetRead("example.xml")  
  ==> t  
  
axlSpreadsheetGetStyles()  
  ==> (("Default" "DEFAULT") ("Title" "TITLE") ("Data" "DATA"))  
  
axlSpreadsheetClose()  
  ==> t
```

### **See Also**

[axlSpreadsheetSetWorksheet](#)

## **axlSpreadsheetGetWorksheets**

```
axlSpreadsheetGetWorksheets()  
)  
==> l_worksheets / nil
```

### **Description**

Retrieves a list of all the worksheets defined in the active spreadsheet. If no worksheets currently exists, nil is returned.

### **Arguments**

none

### **Value Returned**

<i>l_worksheets</i>	List of worksheet names (as ordered in the spreadsheet).
nil	No worksheets current defined / no spreadsheet active.

### **Example**

The following example reads a spreadsheet into memory, then gets the list of defined worksheets in the file.

```
axlSpreadsheetRead("example.xml")  
==> t  
  
axlSpreadsheetGetWorksheets()  
==> ("First" "Second")  
  
axlSpreadsheetClose()  
==> t
```

### **See Also**

[axlSpreadsheetSetWorksheet](#)

## **axISpreadsheetGetWorksheetSize**

```
axISpreadsheetGetWorksheetSize()  
  ==> l_rowsColumns/nil
```

### **Description**

Return the "size" of the current worksheet, in terms of the highest row and column which have data.

### **Arguments**

none

### **Value Returned**

l_rowsColumns	(maxRow, maxColumn).
nil	No worksheets current defined / no spreadsheet active.

### **Example**

The following example reads a spreadsheet into memory, then gets the size of the workbook named "First" before closing the file.

```
axISpreadsheetRead("example.xml")  
  ==> t  
  
axISpreadsheetSetWorksheet("First")  
  ==> t  
  
axISpreadsheetGetWorksheetSize()  
  ==> (5 5)  
  
axISpreadsheetClose()  
  ==> t
```

### **See Also**

[axISpreadsheetSetWorksheet](#)

## **axlSpreadsheetInit**

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

### **Description**

Initializes an empty spreadsheet document to begin filling it with worksheets, styles, and cell data. A new spreadsheet, when first initialized, does not include any of this information. It is completely empty.

If there is a spreadsheet already active in memory, it will be closed. Only one spreadsheet may be active at a time.

### **Arguments**

none

### **Value Returned**

t	Spreadsheet successfully initialized.
nil	Unable to initialize empty spreadsheet. Reason printed to console.

### **Example**

The following example creates a simple spreadsheet, adds information to the first cell ("Hello"), writes the spreadsheet, and closes it.

```
axlSpreadsheetInit()  
==> t  
  
axlSpreadsheetSetWorksheet("First")  
==> t  
  
axlSpreadsheetDefineCell(1 1 "Default" "String" "Hello")  
==> t  
  
axlSpreadsheetWrite("example.xml")  
==> t  
  
axlSpreadsheetClose()  
==> t
```

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

#### **See Also**

[axISpreadsheetClose](#), [axISpreadsheetRead](#), [axISpreadsheetWrite](#)

## **axlSpreadsheetRead**

```
axlSpreadsheetRead(  
    t_fileName  
)  
==> t / nil
```

### **Description**

Read a spreadsheet file on disk into memory for data access and manipulation. File is expected to be in Microsoft XML open spreadsheet format. For text-delimited files, use [axlSpreadsheetReadDelimited](#).

### **Arguments**

t\_fileName                  Name of spreadsheet file on disk to be read.

### **Value Returned**

t	Spreadsheet successfully read; ready for querying.
nil	Unable to read spreadsheet file.

### **Example**

The following example reads a spreadsheet into memory, after which it can be queried for cells' contents, and finally closed.

```
axlSpreadsheetRead("example.xml")  
==> t  
  
...  
axlSpreadsheetGetCell(1 1)  
...  
axlSpreadsheetClose()  
==> t
```

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

#### **See Also**

[axlSpreadsheetClose](#), [axlSpreadsheetInit](#), [axlSpreadsheetWrite](#),  
[axlSpreadsheetReadDelimited](#)

## **axlSpreadsheetReadDelimited**

```
axlSpreadsheetReadDelimited(t_fileName, t_delimiter)
  ==> t / nil
```

### **Description**

Read a text file on disk into memory for data access and manipulation as a spreadsheet. File is expected to be a delimited text file, with cell delimiters as specified in the `t_delimiter` argument. For XML spreadsheets, use [axlSpreadsheetRead](#).

### **Arguments**

<code>t_fileName</code>	Name of text file on disk to be read.
<code>t_delimited</code>	Delimiter character used to separate "cells" in text file.

### **Value Returned**

<code>t</code>	File successfully read; ready for querying.
<code>nil</code>	Unable to read spreadsheet file.

### **Example**

The following example reads a csv file into memory, after which it can be queried for cells' contents, and finally closed.

```
axlSpreadsheetRead("example.txt", ",")
  ==> t

...
axlSpreadsheetGetCell(1 1)

...
axlSpreadsheetClose()
  ==> t
```

### **See Also**

[axlSpreadsheetRead](#)

## **axlSpreadsheetSetCell**

```
axlSpreadsheetSetCell(  
    x_row  
    x_col  
)  
==> t / nil
```

### **Description**

Make the active row/column of the current worksheet active.

### **Arguments**

x_row	Row index (1-based) of cell to activate.
x_col	Column index (1-based) of cell to activate.

### **Value Returned**

t	Cell successfully activated.
nil	Cell not activated. See console for reason.

### **Example**

The following example sets the active cell to be cell 1,1 in the active worksheet.

```
axlSpreadsheetSetCell(1 1)  
==> t  
  
/**/  
/*INDENT ON*/  
  
list axlSpreadsheetSetCell(list l_row, list l_col)  
{  
    long status = SUCCESS;  
    if(sgp_doc && sgp_worksheet)  
    {  
        int row = 0;  
        int col = 0;  
        row = ilGetInt(l_row);  
        col = ilGetInt(l_col);  
  
        if(row > 0 && col > 0)
```

## Allegro SKILL Reference

### Microsoft Excel Integration Functions

---

```
{  
    excelCell* p_cell = excelCellFind(sgp_worksheet, row, col);  
    if(!p_cell)  
    {  
        p_cell = excelCellDefine(sgp_worksheet, row, col, "Default",  
        "String", "");  
    }  
    if(p_cell)  
    {  
        sgp_cell = p_cell;  
    }  
    else  
    {  
        status = ICPEXCELMMSG_SKILL_NOT_DEFINED_1;  
        icp_messagePrint(ICP_MESSAGE_CONSOLE, status, "cell");  
    }  
}  
else  
{  
    status = ICPEXCELMMSG_SKILL_BAD_CELLPOS_2;  
    icp_messagePrint(ICP_MESSAGE_CONSOLE, status, row, col);  
}  
}  
else  
{  
    status = ICPEXCELMMSG_SKILL_NOT_ACTIVE_1;  
    icp_messagePrint(ICP_MESSAGE_CONSOLE, status, "worksheet");  
}  
return(SUCCESS == status ? ilcT : ilcNil);  
}  
  
/*INDENT OFF*/  
/*-  
#ifdef DOC_C
```

#### See Also

[axlSpreadsheetGetCell](#), [axlSpreadsheetSetCellProp](#), [axlSpreadsheetDefineCell](#)

## **axlSpreadsheetSetCellProp**

```
axlSpreadsheetSetCellProp(  
    t_propName  
    t_propVal  
)  
==> t / nil
```

### **Description**

Sets a property on the active cell in the spreadsheet.

### **Arguments**

t_propName	Property to set. Allowable values are: STYLE, TYPE, FORMULA, or VALUE
t_propVal	Value to set this property to.

### **Value Returned**

t	Cell property successfully set.
nil	Property not set (no active cell or invalid property). See console for further details.

Values supported for:

- STYLE – String value giving the ID of style already defined in active spreadsheet.
- TYPE – Number, DateTime, Boolean, String, or Error.
- FORMULA – Any string representing a properly-formatted spreadsheet formula for evaluation.
- Formulas are NOT VERIFIED by this interface for correctness.
- VALUE – Any string providing contents of the cell.

### **Example**

Following example sets the contents of the active cell to the string "New Value".

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

```
a xlSpreadsheetSetCell(1 1)
  ==> t

a xlSpreadsheetSetCellProp("VALUE" "New Value")
  ==> t
```

#### **See Also**

[axlSpreadsheetSetCell](#), [axlSpreadsheetGetCell](#), [axlSpreadsheetDefineCell](#)

## **axlSpreadsheetSetColumnProp**

```
axlSpreadsheetSetColumnProp (
    x_column
    t_propName
    t_propVal
)
==> t / nil
```

### **Description**

Sets a property for the given column of the active worksheet.

### **Arguments**

x_column	Column index to set property for.
t_propName	Property to set. Allowable values are: AUTO_WIDTH, WIDTH, and STYLE.
t_propVal	Value to set this property to.

Values supported for:

- AUTO\_WIDTH – Boolean value (0 or 1).
- WIDTH – Positive integer value in font points.
- STYLE – Style ID name currently defined in active document.

These statements, along with any others, are NOT evaluated by the SKILL API. They will be evaluated by the spreadsheet tool which opens the document. That tool may treat either the AUTO\_WIDTH or the WIDTH tag as having priority of evaluation at its discretion, for instance. The SKILL API will NOT evaluate the AUTO\_WIDTH or other instructions.

### **Value Returned**

t	Property set on column.
nil	Property not set. Reason printed to console.

### **Example**

The following example sets the AUTO\_WIDTH attributed on column 1 of the active worksheet.

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

```
axlSpreadsheetSetRowProp(1 "AUTO_WIDTH" "1")
```

#### **See Also**

[axlSpreadsheetSetRowProp](#)

## **axlSpreadsheetSetDocProp**

```
axlSpreadsheetSetDocProp(  
    t_propName  
    t_propVal  
)  
==> t / nil
```

### **Description**

Sets a property on the document (spreadsheet) itself.

### **Arguments**

t_propName	Property to set. Allowable values are: AUTHOR, LAST_AUTHOR, DATE, COMPANY, or VERSION.
t_propVal	Value to set this property to.

### **Value Returned**

t	Document property successfully set.
nil	Property not set (no active spreadsheet or invalid property). See console for further details.

### **Example**

The following example sets the Author's name for this spreadsheet to be "John Doe".

```
axlSpreadsheetInit()  
==> t  
  
axlSpreadsheetSetDocProp("AUTHOR" "John")  
==> t  
  
axlSpreadsheetSetDocProp("AUTHOR" "Doe")  
==> t  
  
axlSpreadsheetWrite("example.xml")  
==> t  
  
axlSpreadsheetClose()  
==> t
```

## **axlSpreadsheetSetRowProp**

```
axlSpreadsheetSetRowProp(
    x_row
    t_propName
    t_propVal
)
==> t / nil
```

### **Description**

Sets a property for the given row of the active worksheet.

### **Arguments**

x_row	Row index to set property for.
t_propName	Property to set. Allowable properties are, AUTO_HEIGHT, HEIGHT, and STYLE.  Values supported for these properties are:  AUTO_HEIGHT — Boolean value (0 or 1) HEIGHT — Positive integer value in font points STYLE — Style ID name currently defined in active document
t_propVal	Value to set this property to.  AUTO_HEIGHT — Boolean value (0 or 1) HEIGHT — Positive integer value in font points STYLE — Style ID name currently defined in active document

These statements, along with any others, are NOT evaluated by the SKILL API. They will be evaluated by the spreadsheet tool which opens the document. That tool may treat either the AUTO\_HEIGHT or the HEIGHT tag as having priority of evaluation at its discretion, for instance. The SKILL API will NOT evaluate the AUTO\_HEIGHT or other instructions.

## Value Returned

- t                   Property set on row.
- nil                Property not set. Reason printed to console.

## Example

The following example set the auto fit attributed on row 1 of the active worksheet.

```
axlSpreadsheetSetRowProp(1 "AUTO_HEIGHT" "1")
```

## See Also

[axlSpreadsheetSetColumnProp](#)

## **axISpreadsheetSetStyle**

```
axISpreadsheetSetStyle(  
    t_id  
    t_name  
) ==> t / nil
```

### **Description**

Defines or activates the specified style in the active spreadsheet. Styles may be referenced in any worksheet of the spreadsheet. You do not need to redefine the style for each new worksheet you create.

### **Arguments**

t_id	The spreadsheet ID for this style.
t_name	The user "name" for this style / nil. This is the name that is displayed for this style in the Excel style editor and selection pull-down.

### **Value Returned**

t	Style successfully activated / defined.
nil	Style not activated. Reason written to console.

### **Example**

The following example activates the Default style in the active spreadsheet and sets its vertical alignment style to Top-justified.

```
axISpreadsheetSetStyle("Default"  nil)  
    ==> t  
  
axISpreadsheetSetStyleProp("Alignment" "Vertical" "Top")  
    ==> t
```

### **See Also**

[axISpreadsheetSetCell](#), [axISpreadsheetSetWorksheet](#)

## **axlSpreadsheetSetStyleBorder**

```
axlSpreadsheetSetStyleBorder(
    t_position
    t_color
    t_lineStyle
    t_weight
) ==> t / nil
```

### **Description**

Sets the cell border properties for a active style definition.

### **Arguments**

t_position	Position must be one of the accepted Microsoft positions (Left, Right, Top, Bottom, etc).
t_color	Microsoft color name or RGB value (for example, "BLACK" or #FF00AA).
t_lineStyle	Line style to use (normally "Continuous" for a solid line).
t_weight	The thickness of the line, in font points. Must be a positive integer.

Following table lists the values supported for different arguments.

<b>Argument</b>	<b>Supported Values</b>
POSITION	Left, Top, Right, Bottom, DiagonalLeft, or DiagonalRight
COLOR	RGB Value in format "#RRGGBB" or color name from pre-defined names table
LINE_STYLE	None, Continuous, Dash, Dot, DashDot, DashDotDot, SlashDashDot, or Double
WEIGHT	Positive integer value in font points

## Value Returned

t	Border style successfully set.
nil	Border style not set (no active style or invalid parameters). See console for further details.

## Example

The following example defines a new style, "Second", which inherits its settings from the "Default" style, but with a thin border defined.

```
a xlSpreadsheetSetStyle("Second" "Second Style")
  ==> t

a xlSpreadsheetSetStyleParent ("Default")
  ==> t

a xlSpreadsheetSetStyleBorder ("Left"    nil "Continuous" "2")
  ==> t

a xlSpreadsheetSetStyleBorder ("Right"   nil "Continuous" "2")
  ==> t

a xlSpreadsheetSetStyleBorder ("Top"     nil "Continuous" "2")
  ==> t

a xlSpreadsheetSetStyleBorder ("Bottom"  nil "Continuous" "2")
  ==> t
```

## See Also

[axlSpreadsheetSetStyle](#), [axlSpreadsheetSetStyleProp](#), [axlSpreadsheetSetStyleParent](#),  
[axlSpreadsheetGetRGBColorString](#)

## **axISpreadsheetSetStyleParent**

```
axISpreadsheetSetStyleParent(  
    t_parent)  
==> t / nil
```

### **Description**

Sets the active style's parent. Style will inherit default properties from its parent style, therefore only changes need to be specified in the child style. Parent must already be defined for spreadsheet.

### **Arguments**

t_parent	Style ID of parent to link to. Note that the parent must already be defined before it can be referenced by children.
----------	--

### **Value Returned**

t	Style parent successfully set.
nil	Parent not set (no active style or parent style doesn't exist). See console for further details.

**Note:** PARENT value must be a style ID defined in the active spreadsheet document.

### **Example**

The following example defines a new style, "Second", which inherits its settings from the "Default" style, but with text centered in the cell.

```
axISpreadsheetSetStyle("Second" "Second Style")  
==> t  
  
axISpreadsheetSetStyleParent("Default")  
==> t  
  
axISpreadsheetSetStyleProp("Alignment" "Horizontal" "Center")  
==> t
```

### **See Also**

[axISpreadsheetSetStyle](#), [axISpreadsheetSetStyleBorder](#), [axISpreadsheetSetStyleProp](#)

## **axlSpreadsheetSetStyleProp**

```
axlSpreadsheetSetStyleProp(  
    t_type  
    t_propName  
    t_propVal  
) ==> t / nil
```

### **Description**

Sets a specific style property in the active style definition.

### **Arguments**

t_type	Type of property being set. Must be one of: ALIGNMENT, FONT, FILL, NUMBER_FORMAT, PROTECTION.
t_propName	Name of the property being set (varies by type).
t_propVal	Value to set the property to.

### **Value Returned**

t	Style attribute successfully set.
nil	Attribute not set (no active style or invalid parameters). See console for further details.

ALIGNMENT properties and values supported are listed in the following table.

<b>Property</b>	<b>Values Supported...</b>
Horizontal	Automatic, Left, Center, Right, Fill, Justify, CenterAcrossSelection, JustifyDistributed, or Distributed.
Indent	Positive integer value in characters widths to indent.
ReadingOrder	RightToLeft, LeftToRight, or Context
Rotate	Rotation angle in degrees
ShrinkToFit	Boolean value (0 or 1).

**Allegro SKILL Reference**  
Microsoft Excel Integration Functions

---

<b>Property</b>	<b>Values Supported...</b>
Vertical	Automatic, Top, Bottom, Center, Justify, JustifyDistributed, or Distributed
VerticalText	Boolean value (0 or 1).
WrapText	Boolean value (0 or 1).

**Table 27-1 FONT properties and values supported**

<b>Property</b>	<b>Values Supported...</b>
Bold	Boolean value (0 or 1).
CharSet	Integer value
Color	RGB Value in format "#RRGGBB" or color name from pre-defined names table
Family	Automatic, Decorative, Modern, Roman, Swiss, or Script.
Italic	Boolean value (0 or 1).
Outline	Boolean value (0 or 1).
Shadow	Boolean value (0 or 1).
Size	Positive integer value in font size points..
StrikeThrough	Boolean value (0 or 1).
Underline	None, Single, Double, SingleAccounting, or DoubleAccounting.
VerticalAlign	None, Subscript, or SuperScript

**Table 27-2 FILL properties and values supported**

<b>Property</b>	<b>Values Supported...</b>
Color	RGB Value in format "#RRGGBB" or color name from pre-defined names table

**Allegro SKILL Reference**  
Microsoft Excel Integration Functions

---

**Table 27-2 FILL properties and values supported**

<b>Property</b>	<b>Values Supported...</b>
Pattern	None, Solid, Gray75, Gray50, Gray25, Gray125, Gray0625, HorzStripe, VertStripe, ReverseDiagStripe, DiagStripe, DiagCross, ThickDiagCross, ThinHorzStripe, ThinVertStripe, ThinReverseDiagStripe, ThinDiagStripe, ThinHorzCross, or ThinDiagCross.
PatternColor	RGB Value in format "#RRGGBB" or color name from pre-defined names table

**Table 27-3 NUMBER\_FORMAT properties and values supported**

<b>Property</b>	<b>Values Supported...</b>
Format	String defining format of number in cell

**Table 27-4 PROTECTION properties and values supported**

<b>Property</b>	<b>Values Supported...</b>
Protected	Boolean value (0 or 1).
HideFormula	Boolean value (0 or 1).

## Examples

The following example defines a new style, "Second", which inherits its settings from the "Default" style, but with text centered in the cell.

```
axlSpreadsheetSetStyle("Second" "Second Style")
    ==> t

axlSpreadsheetSetStyleParent("Default")
    ==> t

axlSpreadsheetSetStyleProp("Alignment" "Horizontal" "Center")
    ==> t
```

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

#### **See Also**

[axlSpreadsheetSetStyle](#), [axlSpreadsheetSetStyleBorder](#), [axlSpreadsheetSetStyleParent](#),  
[axlSpreadsheetGetRGBColorString](#)

## **axISpreadsheetSetWorksheet**

```
axISpreadsheetSetWorksheet(  
    t_name)  
==> t / nil
```

### **Description**

Makes the specified worksheet the active one for future cell references. If the worksheet does not exist, it is created as the new last worksheet in the document.

### **Arguments**

t\_name                   The name of the worksheet to activate.

### **Value Returned**

t                           Worksheet successfully activated / defined.

nil                       Worksheet not activated. Reason written to console.

### **Examples**

The following example activates the worksheet named "First" in the active spreadsheet, then sets the first column to have a width of 500.

```
axISpreadsheetSetWorksheet("First")  
    ==> t  
axISpreadsheetSetColumnProp(1 "Width" "500")  
    ==> t
```

### **See Also**

[axISpreadsheetSetCell](#), [axISpreadsheetSetStyle](#)

## axlSpreadsheetWrite

```
axlSpreadsheetWrite(  
    t_fileName  
)==> t / nil
```

### Description

Write the spreadsheet in memory to file on disk. File will be written compliant with Microsoft's open spreadsheet XML format.

### Arguments

t_fileName	Name of file to be written to, including path if not to be written to current working directory.
------------	--

### Value Returned

t	File successfully written.
---	----------------------------

### Examples

- The following example creates a simple spreadsheet, adds information to the first cell ("Hello"), writes the spreadsheet, and closes it.

```
axlSpreadsheetInit()  
    ==> t  
  
    axlSpreadsheetSetWorksheet("First")  
        ==> t  
  
        axlSpreadsheetDefineCell(1 1 "Default" "String" "Hello")  
            ==> t  
  
        axlSpreadsheetWrite("example.xml")  
            ==> t  
  
    axlSpreadsheetClose()  
        ==> t
```

**Note:** Specifying file extension other than \*.xml creates the file, but its format remains Microsoft's open spreadsheet XML.

## **Allegro SKILL Reference**

### Microsoft Excel Integration Functions

---

**Note:**

**See Also**

[axISpreadsheetClose](#), [axISpreadsheetInit](#), [axISpreadsheetRead](#)

**Allegro SKILL Reference**  
Microsoft Excel Integration Functions

---

---

# **Plugin Functions**

---

## **Overview**

This chapter describes the AXL-SKILL functions related to plugin APIs.

The axIDll family of APIs provides the ability to bind compiled DLL (shared library) packages into programs supporting the "axl" SKILL APIs. Any publicly exported functions from a DLL can be imported. The only restriction is that exported DLL functions must support the API specified below.

Any reference to 'DLL' is the equivalent to a shared library for UNIX and Linux programmers. All UNIX references apply to Linux.

Creating a plugin requires two components:

- SKILL code to import and wrap the functionality provided by the DLL.
- The DLL/shared library implementing the plugin functionality.

## **SKILL Programming**

The SKILL programming model for plugins is:

- Locate and open a plugin via `ax1DllOpen`. We suggest assigning a returned handle to a global SKILL symbol. On subsequent calls into your SKILL application, the symbol will be non-nil, so the call to `ax1DllOpen` can be skipped. (See `ax1DllOpen`.)
- Import the required symbols from the plugin via `ax1DLLSym`. Like the handle returned by `ax1DllOpen`, these handles should be assigned to global symbols.
- Use either `ax1DllCall` or `ax1DllCallList` to access the capability from the plugin.
- The I/O data types that are supported are documented in `ax1DllCall`.

## DLL Programming

DLL programming allows the use of external developers to utilize existing C/C++ code or develop new capabilities in C/C++ to plugin as extensions to SPB software. SPB software supporting this capability are those that incorporate the "axl" extension package to SKILL.

Implementation of a plugin DLL:

- On both Windows and Linux, all SPB applications are 64-bit applications, which means that any DLL (shared library on Linux) must be built as a 64-bit loadable library. If the DLL is built as a 32-bit library, it will not load successfully. On Windows you must build with the x64 architecture. It is also recommended to build with the release configuration on Windows, rather than the debug configuration. Also, in Visual Studio on Windows in the Project Properties, Linker section's Advanced tab, you must set the option Target Machine to MachineX64 (/MACHINE:X64 option).
- Obtain the compiler environment required by Cadence. (See the Allegro platform documentation.)
- Use the existing Cadence Makefile (UNIX) or Project file (Microsoft) which contains the required compile/link options. You can use these files to build your plugin to adapt your own software configuration environment. (See <cdsroot>/share pcb/examples/skill/plugin.)
- Ensure that the exported DLL functions meet the API requirements. (See API plugin functions below). The include file, <cdsroot>/share pcb/include/axlplugin.h, has the required structure definitions and defines.
- Build your plugin.
- Test it.

Required Cadence files for plugin programming:

<cdsroot>/share pcb/include/axlplugin.h

## API Plugin Functions

The C API for any plugin function is as follows:

```
long <function>(AXLPluginArgs *output, AXLPluginArgs *input)
```

The API takes identical structures for both its input and output arguments. The calling Cadence software updates the input structure with the data passed by the calling SKILL code. It also initializes the output structure.

The Plugin function must return a value and optionally update the output structure with return data (`argv` and `count`). The plugin return value is passed back to the calling SKILL code as follows:

<code>return == 0</code>	Returns a list of output data structure up to the count value. The length of the list is given by 'count'. If 'count' in output structure is 0, the returned list is empty.
<code>return != 0</code>	Return value is returned to SKILL as an integer.

### **AXLPluginArgs Input/Output Structure Members**

<code>version = AXLPLUGIN_VERS_IN</code>	Version of input structure. If additional capability is added in the future, this version number will be bumped. It informs the plugin of the capability supported in the structure.
<code>flag = 0</code>	Future use
<code>maxEntries = AXLPLUGIN_MAX</code>	Can be ignored on input. On output, you cannot exceed this value for return arguments.
<code>count = &lt;number of argv entries&gt;</code>	Number of entries in <code>argv</code> . Basically, the number of arguments provided to the <code>axlDLLCall</code> APIs. Will never be more than <code>maxEntries</code> .
<code>argv[ ]</code>	Array of <code>AXLPluginEntry</code> arguments

### **AXLPluginEntry (argv) Structure Members**

(See Input/Output Data Primitives.)

<code>type</code>	Indicates type of data
<code>data</code>	Union of primitive types supported

For the called DLL function to return data to calling SKILL code, both the `count` and `argv` entries should be set in the output data structure. For each `argv` entry returned, you should also set the data type to one of the primitives shown below. Under no situation should you attempt to return more than `maxEntries`.

If your exported DLL functions will be used outside your programming environment, then you should valid check the input arguments either in the SKILL wrapper or in your DLL function.

For example, if you expect a string argument and the calling SKILL code passes an integer, a program exception will occur if the data is not checked.

## Input/Output Data Primitives

A set of I/O data primitives is supported. The `argv` entry of `AXLPluginArgs` is an array of these structures. Each array member has its data type indicated and the data itself. Both the input and output data use the `AXLPluginArgs` structure.

The data primitives that are supported are:

Type	SKILL type	C type	C struct member
AP_BOOL	t/nil	int	b_value
AP_LONG	x_value	long	l_value
AP_DOUBLE	f_value	double	d_value
AP_CONST_STRING	t_value/s_value	char *	cs_value
AP_STRING	t_value	char *	s_value (output only)
AP_XY	f_value:f_value	AXLXY	xy_value

AP\_STRING types will not be seen on input. All strings passed from Allegro to the plugin are AP\_CONST\_STRING. To return a string, the plugin may use either AP\_CONST\_STRING or AP\_STRING. If AP\_STRING is used, then Allegro will free the memory associated with the string using the standard C "free" API, which means you must allocate it via `malloc`.

Other data type restrictions are:

- The plugin must not modify in place any input AP\_CONST\_STRING; corruption of SKILL will occur.
- To maintain AP\_CONST\_STRING input data across function calls, you should make a copy of them. SKILL may garbage collect the memory after your DLL function returns.
- If you use your own memory manager, do not use AP\_STRING types for return. These require the use the standard `malloc` memory manager.
- Input arguments can be SKILL symbols ('<symbol>') but these are converted to AP\_CONST\_STRING types.
- The STRING types do not support wide character sets; use only characters in the ASCII character set.

- AP\_XY represents a (x y) location and is implemented as a structure of two doubles.
- If AP\_BOOL type is used, two defines are provided in `axlplugin.h`:
  - AXLPLUGIN\_TRUE = 1
  - AXLPLUGIN\_FALSE = 0

Return value from plugin exported back to SKILL:

- If `output->count` is 0, then we look at the return value from the plugin function and return `x_value` to symbol.
- Else (non-zero count), we return this value to SKILL as a SKILL integer type.

## Programming Restrictions, Cautions and Hints

- Cadence only supports the compiler listed in the SPB Platform documentation. Also required are any compiler and DLL linker options listed.
- If your DLL has dependencies upon other DLLs, make sure those do not conflict with DLLs used by Cadence. Typically, they need to be the same version and not built with debug options.

To determine the DLLs used by Cadence use:

- Windows: depends or procexp ([www.sysinternals.com](http://www.sysinternals.com))
- Linux: ldd <program name>
- On Windows, you can include UI components in your plugin. This capability is not supported on Linux.
- Wide character types are not supported in the plugin to the Allegro interface.
- On Linux, if threading is used, then you should use POSIX threads (pthreads).
- STL programming should use the default STL provided by the Cadence required compiler. Do not use a third party STL.
- On Windows, if DLL is MFC based, then do not compile it debug. MFC Classes change in size if code is built debug and are not compatible with non-debug MFC code. There are methods described on the Web on how to build MFC debuggable without requiring the MFC debug shared library. Typically on Windows, Debug DLLs are not compatible with non-Debug DLLs. Also many different versions of MFC exist which are incompatible with one another. Use one of the binary query tools (see above) to determine the version of MFC currently required by Cadence.

- There are currently no methods to make function calls back to Allegro from the plugin.
- Your DLL can be used across multiple Cadence releases without recompiling your plugin. Re-compiling is only required if Cadence changes the compiler in the release.
- Programming errors in your plugin can crash the host program. In rare cases, these bugs can corrupt an Allegro database. (See customer support below.)
- On Windows, to access `stdout/stderr`, set the `TELCONSOLE=1` as the OS level. This variable cannot be set via Allegro's `env` file. This is also considered a debug environment, so it should not be used during board design.

## Performance Considerations

The following issues should be considered if high performance is required:

- Locating and loading a plugin. This should be done once per activation. Assign the handle return by `axlDllOpen` to a global symbol to prevent SKILL garbage collection. Do not close the plugin.
- Import symbols of a plugin once. The `axlDll` interface internally caches symbol import so subsequent lookups will be faster than the initial call.
- A plugin function should do meaningful work since there is some overhead converting input data from SKILL to native C types and doing the opposite for return data.

## Cadence Customer Support

Cadence does not provide support for debugging customer-developed plugins.

An environment variable, `allegro_noextension_plugin`, is available to disable plugin capability. If Allegro starts crashing or databases become corrupt, you should set this variable to determine if a plugin is the cause.

## Examples

An example containing SKILL, the plugin C code, Gnu Makefile (Linux) and project file (Visual Studio) is contained at `<cdsroot>/share/pcb/examples/plugin`. In addition, a pre-built plugin module is contained in the Cadence install hierarchy so you do not need to compile and link the plugin source code to run the SKILL code.

The example plugin provided has three exported symbols:

- An echo API, which returns as output any input given.

## **Allegro SKILL Reference**

### Plugin Functions

---

- A `x_value` API, which returns the number of input arguments.
- A distance implementation to calculate the distance between two points.

## axIDllCall

```
axIDllCall(  
    o_pluginFunc  
    [g_arg1]  
    ...  
    [g_arg10]  
) -> nil/x_value/lg_data
```

### Description

This function calls a symbol that has been imported from a plugin. As the first argument, it requires `o_pluginFunc`, which was returned via a call to `axIDllSym`. The rest of the arguments are what the implement plugin API has defined.

The return from the call is one of the following:

- `nil`: error processing data
- `x_value`: plugin function returned a non-zero result
- `lg_data`: plugin function returned a zero and its output values are returned in a list

In turn it calls the import function from the plugin.

### Arguments

<code>o_pluginFunc</code>	plugin symbol handle
<code>[g_arg1..10]</code>	up to 10 arguments

### Value Returned

<code>nil</code>	Error in processing arguments or funding functions.
<code>x_value</code>	If plugin function returns a non-zero, then this is what is returned.
<code>lg_data</code>	If plugin function returns zero, then its output arguments are processed and returned as a list. If the output argument list from the plugin has 0 entries, then an empty list is returned.

**See Also**

[axlDllOpen](#)

**Example**

Example carried from axlDllSym

```
axlDllCall(_ashDistance 10:0 25.1:0) -> 15.1
```

## **axIDllCallList**

```
axIDllCallList(  
    o_pluginFunc  
    l_args/nil  
) -> nil/x_value/lg_data
```

### **Description**

This function is identical to `axIDllCall` except it takes a list of arguments to pass to the plugin function. Unlike `axIDllCall`, which is limited to 10 arguments, this interface can take up to 512 arguments.

See `axIDllCall` for a further explanation.

### **Arguments**

`o_pluginFunc`      Plugin symbol handle

`l_args`              A list of up to 512 arguments

### **Value Returned**

`nil`              Error in processing arguments or funding functions.

`x_value`              If plugin function returns a non-zero, then this is what is returned.

`lg_data`              If plugin function returns zero, then its output arguments are processed and returned as a list. If the output argument list from the plugin has 0 entries, then an empty list is returned.

### **See Also**

[axIDllCall](#), [axIDllOpen](#)

### **Example**

From `axIDllOpen` example

```
_ashEcho = axIDllSym(_ashTestDll "ashEcho")  
axIDllCallList(_ashEcho list(-1 -2.0 nil "another string"  
    -10.1:-2 0.2))
```

## **ax1DllClose**

```
ax1DllClose(  
    o_plugin  
)  
==> t/nil
```

### **Description**

This closes an open plugin handle. Once a handle is closed, you can no longer call functions obtained from the plugin. Also, a plugin is automatically closed when there are no active references to it via SKILL's garbage collection.

It is not advisable to close a plugin due to performance considerations.

### **Arguments**

`o_plugin`      Plugin handle obtained from `ax1DllOpen`.

### **Value Returned**

`t` if successful to close, `nil` if handle is not a legal handle

### **See Also**

[ax1DllOpen](#)

### **Example**

See example referenced by `ax1DllOpen`.

## **ax1DllDump**

```
ax1DllDump(  
)  
==> l_dllLoad/nil
```

### **Description**

This is a debug function that reports all plugins loaded by SKILL.

### **Arguments**

None

### **Value Returned**

List of plugin handles or `nil` if no loaded plugins.

### **See Also**

[ax1DllOpen](#)

### **Example**

```
ax1DllDump()
```

## axlDllOpen

```
axlDllOpen(  
    t_dllname  
)  
==> o_plugin/nil
```

### Description

This binds a DLL/shared library to the current program. While this can load any DLL, only those built to be compatible with the axl Plugin model can be utilized via SKILL (see [DLL Programming](#) on page 1612 for information on building compatible DLLs).

If the DLL name does not have a directory path component, then *AXLPLUGINPATH* environment variable is used to search for the DLL.

After a DLL is successfully loaded, you need to import one or more symbols (axlD11Sym).

### Plugin Attributes

---

Name	Type	Description
name	string	Name of plugin file (DLL name)
functions	I_dbid	Disembodied property list name/value pairs of imported symbols (t_name o_pluginFunc)
objType	string	"plugin"

**Note:** To access imported plugin function types do a

```
<o_plugin>->functions-><pluginFuncName>
```

### Arguments

t\_dllname      Name of DLL. For platform independence, it is strongly suggested that you do not include the file extension or a directory path component.

### Value Returned

o\_plugin

nil      Can't locate library or not a DLL.

## See Also

[DLL Programming](#) on page 1612, [axlDllSym](#), [axlDllCall](#), [axlDllCallList](#), [axlDllClose](#), [axlDllDump](#)

## Example

Open Cadence test DLL

```
_ashTestDll = axlDllOpen("axlecho_plugin")
```

## **ax1DllSym**

```
ax1DllSym(  
    o_plugin  
    t_symbolName  
)  
==> o_pluginFunc/nil
```

### **Description**

This imports a symbol from a loaded dll. A dll can have one or more exported symbols. The symbol must have been exported from the dll when the dll was compiled and linked (See ax1DllDoc).

#### PluginFunc Attributes

Name	Type	Description
name	string	Name of imported symbol file
functions	nil	Always nil
objType	string	"pluginFunc"

### **Arguments**

o\_plugin            dll handle from ax1DllOpen  
t\_symbolName        Name of an exported function within the dll

### **Value Returned**

o\_pluginFunc        Symbol handle  
nil                  Error; symbol not present in dll.

### **See Also**

[ax1DllOpen](#)

## Example

Load the distance symbol from the axl plugin test dll

```
_ashDistance = axlDllSym(_ashTestDll "ashDistance")
```

---

# Skill Language Extensions

---

## **axldo**

```
axldo(
  g_initList
  g_terminateList
  [g_body]
) -> g_result

axldoStar(
  g_initList
  g_terminateList
  [g_body]
) -> g_result
```

### Description

A do function, modeled after the CL (Common Lisp) do.

Public:

```
(defmacro do ((var [init [step]]) ...) (end-test result result ...) @body)
(defmacro doStar ((var [init [step]]) ...) (end-test result result ...))
```

The do macro provides a generalized iteration facility, with an arbitrary number of *index variables*. These variables are bound within the iteration and stepped in specified ways. They may be used both to generate successive values of interest or to accumulate results. When an end condition is met (as specified by end-test), the iteration terminates, the result sexps are successive evaluated, and the value of the last result sexp is returned.

The first item in the form is a list of 0 or more index-variable specifiers. Each index-variable specifier is a list of the name of the variable, var; an initial value, init; and a stepping form, step.

If init is omitted, it defaults to nil. If step is omitted, the var is not changed by the do construct between repetitions (though code within the do is free to alter the value of the variable by using setq).

An index-variable specified can also be just the name of a variable. In this case, the variable has an initial value of nil and is not changed between repetitions. This would be used much as a locally scoped variable in a let statement would be.

Before the first iteration, all the init forms are evaluated, and each var is bound to the value of its respective init. Because this is a binding, and not an assignment, when the loop terminates the old values of the variables is restored. All of the init forms are evaluated before any var is bound; hence all the init forms may refer to the old binds of all the variables (that is, to the values visible BEFORE beginning execution of the do).

**Note:** All init bindings are done in parallel for `axlndo`, and serially for `axlndoStar`.

The second element of the loop is a list of an end-testing predicate form end-test, and zero or more result forms. This resembles a 'cond' clause. At the beginning of each iteration, after processing the variables, the end-test is evaluated. If the result is nil, execution proceeds with the body of the form. If the result is non-nil, the result forms are evaluated in order as an implicit 'progn', and then the do returns the value of the last evaluated result.

At the beginning of each iteration, the index variables are updated as follows.

- All the step forms are evaluated, from left to right, and the resulting values are assigned to the respective index variables. Any variable that has no step value is not assigned.

## Arguments

<code>g_initList</code>	0 or more index variable specifiers
<code>g_terminateList</code>	end test predicate
<code>g_body</code>	body of procedure

## Value Returned

Value resulting from evaluating last result sexp.

## See Also

[letStar](#)

## copyDeep

```
copyDeep (
    l_object
) -> l_newObject
```

### Description

This function recursively copy a list.

The copy function makes a new list containing copies of all top level elements in the source list. But each new element contains references to the same sublist elements as the source list. Thus, if sublist items are modified in the source list they are also modified in the copy, and vice-versa. The copyDeep function makes a complete copy of the list, top to bottom. So changes in one list do not affect the other. This allocates more memory, of course.

### Arguments

l\_object    The list to be copied

### Value Returned

A new list identical to the original but sharing no memory.

## **isBoxp**

```
isBoxp (  
        g_bBox  
        )  
      ==> t/nil
```

### **Description**

Checks argument to see if it is a valid bounding box. A valid bounding box should be of the form of ((a b) (c d)) where a, b, c, and d are all numbers and a != c and b != d, to ensure that the bounding box encloses some area.

### **Arguments**

g\_bBox      input argument to be tested.

### **Value Returned**

t: If g\_bBox is a valid bounding box.

nil: If g\_bBox is not a valid bounding box.

## **lastelem**

```
lastelem(  
    l_list  
) -> g_elem
```

### **Description**

The `last()` function returns the last LIST object in a list, but `lastelem()` takes the car of that to get the last ATOM.

### **Arguments**

`l_list`      a list

### **Value Returned**

The last atom element of a list.

### **Example**

```
l = list(1 2 3)  
last(l) -> (3)  
lastelem(l) -> 3
```

## letStar

```
letStar(  
    l_bindings  
    [@body]  
) -> g_lastValue
```

### Description

This is a let\* implementation of CL (Common Lisp). This is a procedure function.

### Arguments

l_bindings	list of l_varbind
	l_varbind is (<name> <value>) ■ name: interned as lexically-scoped symbol ■ value: evaluated to arrive at value
@body	one or more expressions to be evaluated.

### Value Returned

last value of @body

### See Also

[axldo](#)

## **listnindex**

```
listnindex(  
    l_theList  
    g_item  
) -> x_found/nil
```

### **Description**

Finds the position of an item in a list. Works just like `nindex`, but finds the position of an element in a list instead of a character in a string. An integer denoting the sequence number of the first matching element is determined.

### **Arguments**

`l_theList`      A list containing the element to be found  
`g_item`        The element to be found

### **Value Returned**

The position in the list where the element was first found, or `nil` if it is not found.

**Note:** `x_found` is 0 based so you use other SKILL functions like `nth`

### **Example**

```
listnindex( '("three" "dog" 'night) "dog") -> 1
```

## **movedown**

```
movedown (
    g_elem
    l_list
) --> l_newlist
```

### **Description**

Find all occurrences of element within a list and moves them one item closer to the list tail.  
No action for other items and the last element of the list.

This destructively modifies the list.

### **Arguments**

g\_elem      The element to be moved, matched using (equal)

l\_list      The list containing the element to be moved.

### **Value Returned**

The modified list.

## **moveup**

```
moveup (
    g_elem
    l_list
) --> l_newlist
```

### **Description**

Moves an element one item closer to the head of a list. Finds all occurrences of the element within a list and moves them one item closer to the list head. No action for other items and the first element of the list.

This destructively modifies the list.

### **Arguments**

g\_elem    The element to be moved, matched using (equal)

l\_list    The list containing the element to be moved.

### **Value Returned**

The modified list.

## **parseFile**

```
parseFile (
    t_fileName
    s_handler
    [t_breakChars]
)
==> g_result
```

### **Description**

Parse all lines of a file. Opens input file and reads it line by line. Each line is parsed using `parseQuotedString`. The result of `parseQuotedString` is passed to the application defined handler `s_handler`. The handler defines the return value.

## Arguments

t_fileName	Name of file to be read.
s_handler	Application callback function to process parsed arguments <pre>s_handler (     l_curLineInfo     l_lineArgs     g_result ) ==&gt; g_result</pre>
l_curLineInfo	List of info which describes the current line being processed. Defined as: <pre>(t_fileName g_lineNo t_curString)</pre> <ul style="list-style-type: none"><li>■ t_fileName: Name of file being read</li><li>■ g_lineNo: Current line number</li><li>■ t_curString: Unparsed file line</li></ul>
l_lineArgs	List of strings that result from parsing the line.
g_result	The application callback result. This is continually passed to s_handler so that a result can be built up from the processing of all file lines. This will be nil the first time and will be whatever s_handler last returned for subsequent calls.
t_breakChars	Optional string containing characters that are used as break characters. The default is "' \\ \" (quote chars and space char).

## Value Returned

g\_result      The application callback result from the last call to s\_handler.

## parseQuotedString

```
parseQuotedString(  
    t_string  
    [t_breakCharacters]  
)  
--> l_strings
```

### Description

Breaks a string into a list of words. Multiple words enclosed within quotation marks are treated as single words.

### Arguments

t_string	String to be parsed.
t_breakCharacters	Optional string containing characters to be used as break characters. The default is " \" " (quote chars and space char).

### Value Returned

l_strings	A list of strings parsed from the t_string argument.
-----------	--

**Note:** A word break is not created at the start of a quote unless the quote character is a break character.

### Examples

- parseQuotedString( "111 222'333 444'" " " ) => ("111" "222333 444")
- parseQuotedString( "111 222'333 444'" "' " ) => ("111" "222" "333 444")

## **pprintln**

```
pprint(  
    g_item  
    [p_port]  
) -> t/nil
```

### **Description**

Prints a newline character at the end of the line.

### **Arguments**

- g\_item      The item to be printed.  
p\_port      The optional parameter that specifies the port to write to. Default value is stdout.

### **Value Returned**

Returns item pretty printed plus a newline character.

## **propNames**

```
propNames (
    g_propList
) -> ls_names/nil
```

### **Description**

Use this command to get the names of properties in a disembodied property list. Walk each property in disembodied property list, building a list of the names of each property.

**Note:** Order of returned property names is unspecified.

### **Arguments**

g\_propList    A disembodied property list.

### **Value Returned**

ls\_names    A list of symbols corresponding to the names of each property found in the list.

### **Example**

```
n = ncons(nil)
n->one = 1
n->two = 2
propNames(n) -> (one two)
```

---

# **Logic Access Functions**

---

## **Overview**

This chapter describes the AXL-SKILL functions related to schematic capture or the logical definition of the design.

## **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, vias, and shapes. Vias may not stay on net assigned if they do not connect to an object on the new net. This is not applicable if the new net has RETAIN\_NET\_ON\_VIAS property.

## 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 can be used as an alternative to a dbid net.
<i>g_riputp</i>	An optional flag to ripup clines and vias connected to modified objects. possible values are: <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> .
<i>g_ignoreFixed</i>	By default, won't allow net assignment if FIXED property is present on the object(s) in question. Setting this argument to <i>t</i> ignores the FIXED property. If <i>g_riputp</i> is <i>t</i> , and connected clines have the FIXED property then the clines will remain and an error message thrown.

## Value Returned

<i>t</i>	At least one object changed net.
<i>nil</i>	No object changed net.

## See Also

[axlDBCreateNet](#), [axlDBIgnoreFixed](#)

## Example:

### ■ Simple re-assignment

```
pin_id->net->name => ""  
net_id->name => "GND"
```

```
axlDBAssignNet(pin_id net_id t)=> t  
pin_id->net->name => "GND"
```

### ■ Interactive exploration; use the ashOne shareware provided in examples area of cdsroot.

## **Allegro SKILL Reference**

### Logic Access Functions

---

```
;select an object
item = ashOne()
new net
net = "net1"

axlDBAssignNet(item net t t)
```

## **axlDBCreateConceptComponent**

```
axlDBCreateConceptComponent (
    s_refdes
    s_partPath
    s_logName
    s_primName
    [s_pptRowName]
)
⇒ r_dbid/nil
```

### **Description**

Given the Concept information needed to describe an Allegro PCB Editor device, create the Allegro 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_prt` 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

<i>s_refDes</i>	Reference designator for the new component.
<i>s_partPath</i>	Full path to the <code>chips_ptr</code> file containing the description of the desired logical part. Determine using the <code>cptListComponentLibraries</code> function.
<i>s_logName</i>	Name of the desired logical part. You can determine this using the <code>cptListComponentPrimitives</code> function.
<i>s_primName</i>	Name of the desired primitive. You can determine this using the <code>cptListComponentPrimitives</code> function.
<i>s_pptRowName</i>	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

<i>r_dbid</i>	<i>dbid</i> of the new Allegro PCB Editor component instance.
<i>nil</i>	Unable to create component instance.

**Note:** The actual name of the resulting Allegro 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.

## Allegro SKILL Reference

### Logic Access Functions

---

## axlDBCreateComponent

```
axlDBCreateComponent(  
    s_refDes  
    s_deviceName  
    [s_package]  
    [s_value]  
    [s_tolerance]  
)  
==> r_dbid/nil
```

### Description

Given the information needed to describe a Allegro PCB Editor device, create the Allegro 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 Allegro 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 Allegro 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.

## Example

Create a new empty comp from an existing one, ashOne can be found at:

```
<cdsroot>/share/pcb/examples/skill/examples/ash-fxf/ashone.il
syminst = ashOne()
ci = syminst->component
nci = axlDBCreateComponent("C1_NEW" ci->deviceType ci->package
ci->compdef->prop->VALUE ci->compdef->prop->TOL)
```

If your component is not a discrete component, you do not need the fourth and fifth arguments, but the above example will work for all components.

## See Also

[axlComponentChangeClass](#)

## **axIDBCreateManyModuleInstances**

```
axIDBCreateManyModuleInstances (
    t_name
    t_moddefName
    x_tileStartNum
    l_origin
    l_offset
    x_num_tiles
    f_rotation
    x_logicMethod
    [l_netExcept]
    [g_mirror]
)
==> 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. <ul style="list-style-type: none"><li>■ 0: No logic.</li><li>■ 1: Logic from schematic.</li><li>■ 2: Logic from module definition.</li></ul>
<i>l_netExcept</i>	Optional list of net names to add to the net exception list.
<i>g_mirror</i>	Optional if modules should be mirrored

## Value Returned

<i>lo_result</i>	If successful, returns the list of database objects that belong to the module instances created.
<i>nil</i>	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`.

## **Allegro SKILL Reference**

### Logic Access Functions

---

## **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**

t	Module definition successfully created.
nil	No module definition created.

### **Example**

```
axlSetFindFilter(?enabled '("noall" "components") ?onButtons '("noall"  
    "components"))  
axlSingleSelectName("COMPONENT" "U1")  
compl = car(axlGetSelSet())  
axlSingleSelectName("COMPONENT" "U2")  
comp2 = car(axlGetSelSet())  
axlDBCreateModuleDef("comps" '(0 0) '(compl 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.
<i>nil</i>	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.

## **axlDBCreateNet**

```
axlDBCreateNet(  
    t_netName  
)  
==> o_dbid/nil
```

### **Description**

Creates a net in database if does not exist or returns `dbid` of net if it exists.

### **Arguments**

*t\_netName*                    Net name to create, or find.

### **Value Returned**

`nil` if not created, or a `axl dbid` of net

### **See Also**

[axlDBAssignNet](#)

### **Example**

```
net = axlDBNetCreate("gnd")  
=> dbid:123456  
net->name  
=> "GND"
```

## **ax1DBCreateSymDefSkeleton**

```
ax1DBCreateSymDefSkeleton (
    l_symbolData
    l_extents
    [l_pinData]
)
==> ax1DBID/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 “skeleton” definition has been created, you add the rest of the symbol geometry with additional ax1DBCreate calls. This provides the ability to create symbols that do not exist in the library.

Shape symbol:

- You may only attach a single shape to a shape symbol, no voids.
- Layer required is “ETCH/TOP”.
- Extents should be larger than the shape but there is no adverse impact if they are significantly larger.

Flash symbol:

- You may attach multiple shapes, but none may contain voids.
- Layer required is “ETCH/TOP”.
- Extents should be larger than the shape but there is no adverse impact if they are significantly larger.

## 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 <code>axlPinData</code> defstructs for the pins.

## Value Returned

`axlDBCreateSymDefSk` nil if not created, or `axlDBID` of the symbol definition.  
eleton

## Examples

```
symdef = axlDBCreateSymDefSkeleton('("shape_pad" "shape") list(-100:-100 100:100))
p = axlPathStart( list(-4:10 4:10 8:0 4:-10 -4:-10 -4:10))
s = axlDBCreateShape(p t "ETCH/TOP" nil symdef)
```

Creates a shape symbol.

```
symdef = axlDBCreateSymDefSkeleton('("flash_pad" "flash") list(-100:-100 100:100))
p = axlPathStart( list(-4:10 4:10 8:0 4:-10 -4:-10 -4:10))
ps = axlPathStart( list(-4:10 4:10 4:-10 -4:-10 -4:10))
s = axlDBCreateShape(p t "ETCH/TOP" nil symdef)
s = axlDBCreateShape(ps t "ETCH/TOP" nil symdef)
```

Creates a flash symbol.

## **ax1DBDummyNet**

```
ax1DBDummyNet (
    g_mode)
-> lo_dbid/nil
```

### **Description**

This command returns all dummy nets in design. Two courtesy options provided are:

- ❑ 'pin for any dummy net that has a pin return the first pin of the dummy net
- ❑ 'shape for any dummy net that has a shape return the first shape of the dummy net

Typically each dummy net in design will only have a single pin or shape but this may not always be the case. Clines or vias cannot, by themselves exist on a dummy. Symbols (`dra`) do not have dummy nets

### **Arguments**

<i>g_mode</i>	Can have following two values: <ul style="list-style-type: none"><li>■ 'pin – returns a list of first pins on dummy nets</li><li>■ 'shape – returns list of first shapes on dummy nets</li></ul>
---------------	--

### **Value Returned**

- `t` - indicates success
- `nil` - failed due to incorrect arguments

### **See Also**

[ax1IsDummyNet](#), [ax1DbidName](#)

### **Examples**

#### **1. Print all 1st pins on dummy nets**

```
foreach( mapc x ax1DBDummyNet('pin) printf("%s\n" ax1DbidName(x)) )
```

## **Allegro SKILL Reference**

### Logic Access Functions

---

#### **2. Get all dummy nets on design**

```
p = axlDBDummyNet(nil)
```

**axIDbidName**

```
ax1DbidName (
```

## Description

Provides the standard Allegro PCB Editor name of a database object. Many of the named Allegro 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 Allegro PCB Editor database id.

**Note:** Some Allegro PCB Editor database ids are pseudo ids (for example, ax1DBGetDesign and pads) and generate a nil return.

## Value Returned

Allegro PCB Editor name of object, or nil if not a dbid or true Allegro 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"
```

## **Allegro SKILL Reference**

### Logic Access Functions

---

## 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/lo_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.
```

You cannot create a differential pair if two nets belong to the same XNet.

## Arguments

<i>o_diffpair</i>	Differential pair <i>dbid</i>
<i>lo_diffpair</i>	List of differential pair dbids (delete mode only)
<i>t_diffpair</i>	Differential pair name (string)
<i>o_net1/o_net2</i>	Net <i>dbid</i> .
<i>t_net1/t_net2</i>	Net name (string)
<i>g_mode</i>	Add to testpoint to top or bottom or clear one

## Value Returned

Values returned depend on whether adding, modifying, or deleting.

<i>o_diffpair</i>	<i>dbid</i> of new or modified differential pair
<i>t</i>	Delete was successful
<i>nil</i>	Error due to incorrect arguments.

## Example 1

```
dbid = axlDiffPair("DP" "NET1+" "NET2-")
```

Creates a differential pair and names it DP1.

## Example 2

```
dbid = axlDiffPair(dbid "NET1+" "NET1-")
```

Modifies a differential pair.

## Example 3

```
axlDiffPair(dbid)
```

Deletes a differential pair.

## Example 4

```
axlDiffPair(axlDBGetDesign() ->diffpair)
```

Delete all differential pairs in design.

## axlDiffPairAuto

```
axlDiffPairAuto(  
    t_diffPairPrefix  
    t_posNetSuffix  
    t_negNetSuffix  
    [g_returnDiffPairList]  
)  
⇒ x_cnt/(xcnt lo_diffpair)/nil
```

### Description

Allows automatic generation of the differential pair. Generates the set of differential pairs based on the provided positive (*t\_posNetSuffix*) and negative (*t\_negNetSuffix*) suffixes used in your net naming scheme.

You may optionally provide a prefix (*t\_diffPairPrefix*) used in generating the differential pair names of the form: <*t\_diffPairPrefix*> + *netname* - *suffix*.

If nets are part of busses and end the bit field syntax (<1>), the bit field portion is ignored when performing suffix matching. If a differential pair is created the bit number is added to the base net name used in forming a differential pair. For example, given two nets; DATA\_P<1> and DATA\_N<1> will result in a differential pair called DP\_DATA1 with this call:

```
axlDiffPairAuto("DP_" "_P" "_N")
```

### Arguments

<i>t_diffPairPrefix</i>	String to prefix differential pair names. Use " " if no prefix is desired.
<i>t_posNetSuffix</i>	Suffixes used to identify + differential pair members.
<i>t_negNetSuffix</i>	Suffixes used to identify - differential pair members.
<i>g_returnDiffPairList</i>	Controls return.

### Value Returned

Returns depend on value of *g\_returnDiffPairList* as shown:

<b>g_returnDiffPairList</b>	<b>Value Returned</b>	<b>Description</b>
nil	<i>x_cnt</i>	Number of differential pairs created.
t	( <i>x_cnt</i> , <i>lo_diffpair</i> )	List of differential pairs created

The default is the `nil` case.

## Examples

### Example nets

Two nets called `NET1+` and `NET1-` are passed to this function.

#### Example 1

```
axlDiffPairAuto("DP_" "+" "-")
```

Shows differential pair creation and name generation. Results in one differential pair called `DP_NET1` with members `NET1+` and `NET1-`.

#### Example 2

```
axlDiffPairAuto("") "+" "-")
```

Given the same inputs as the previous example, will result in a differential pair with the same members, but names the differential pair `NET1`.

## **axlDiffPairDBID**

```
axlDiffPairDBID(  
    t_name  
)  
⇒ o_dbid/nil
```

## Description

Returns the *dbid* of the named differential pair (*t\_name*), if it exists in the database.

## Arguments

*t\_name* Differential pair name.

## Value Returned

*o\_dbid*                            *dbid* of differential pair if it exists.

nil Differential pair does not exist.

## Example

## List differential pairs:

```
axlDBGetDesign() ->diffpair
```

## 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

<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 add to match group.
<i>lo_dbid</i>	List of legal database dbids to add to match group.

### Value Returned

<i>t</i>	Added elements.
<i>nil</i>	Failed one or more element additions.

### See Also

[axlMatchGroupCreate](#)

## **Allegro SKILL Reference**

### Logic Access Functions

---

#### **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)
```

## **axIMatchGroupCreate**

```
axIMatchGroupCreate(  
    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. Use the [axIMatchGroupAdd](#) command to populate the match groups.

**Note:** The `axIMatchGroupCreate` 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.

You can access list of match groups in database by:

```
axIMatchGroupList()
```

### ***RELATIVE\_PROPAGATION\_DELAY***

The `RELATIVE_PROPAGATION_DELAY` property can be added to match groups, xnets, and pinpairs. If you add it to the match group then any match group member that does not have that 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) ....)
```

Thus if a pinpair belongs to 2 match groups, you see two lists that are the inherited. A pinpair in a single match group has a single list of list. For example, a pinpair in MG3 with global scope and an override of delta/tolerance of 10ns:5% reports:

```
rpd = ((dbid:61315360 "MG3:G:::10 ns:5 %"))
```

The same pinpair without an override will report default match group value as

```
rpd = ((dbid:61315360 "MG3:G:AD:AR:0 ns:5 %"))
```

## Allegro SKILL Reference

### Logic Access Functions

---

In both the examples, the dbid references the match group id (MG3).

The RELATIVE\_PROPAGATION\_DELAY syntax is discussed in the Allegro Property Reference Manual. In general the syntax is:

```
<Match group name>:<scope>:<pinpair>:<value>
```

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:

```
axlMatchGroupProp. (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 should not reference an explicit pinpair but, if not empty, should use a pinpair type (for example, "AD:AR","D:R" etc.)

For nets and xnets, if you access the RELATIVE\_PROPAGATION\_DELAY property, you may see the flattened version. This implies that if nets and xnets are part of multiple Match Groups, they all 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. This is present to support legacy applications like netlisters. The RPD property that is dbids for pinpairs, net and xnets breaks this concatenation. Using [axlDBAddProp](#) and [axlDBDeleteProp](#) commands to modify the property may effect all match groups and pinpairs. It is recommended that [axlMatchGroupProp](#) is used 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 :

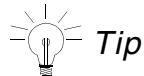
```
mg = axlMatchGroupCreate("mg1")
```

## **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**

<i>o_mgdbid</i>	dbid of a match group.
<i>t_mgName</i>	Name of a match group.

### **Value Returned**

<i>t</i>	Match group deleted.
<i>nil</i>	Failed.

### **See Also**

[axlMatchGroupCreate](#)

### **Examples**

Delete match group created in axlMatchGroupCreate:

```
mg = car(axlSelectByName ("MATCH_GROUP" "MG1"))  
axlMatchGroupDelete(mg)
```

or

```
axlMatchGroupDelete ("MG1")
```

## axlMatchGroupProp

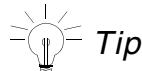
```
axlMatchGroupProp (
    o_mgdbid/t_mgName
    o_dbid
    t_value/nil
) ==> t/nil
```

### 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:

## **Allegro SKILL Reference**

### Logic Access Functions

---

```
axlMatchGroupRemove (mg    car (nets) )
```

## **axlNetSched**

axlNetSched()

==> *t*

### **Description**

This is the main routine that the command processor calls for the `net schedule` command.

### **Arguments**

None

### **Value Returned**

*t*

## 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 unordered 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 ratTs) do not belong to the same XNet.

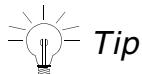
If the pinpair was created in an ECSet, the ECsetDerived attribute will be t and you cannot delete it. You must find the ECSet(s) associated with a net, XNet, differential pair or bus and modify the pinpair in the associated ECset..

At database save, a pinpair must be part of a match group or have a legal electrical constraint property assigned to it, otherwise it will be deleted. 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 attribute as a list of lists. The reason for this is that a pinpair can be a member of multiple matched groups.

See axlMatchGroupCreate for more information.



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.

<i>nil</i>	Error
<i>t</i>	Deletion was successful.
<i>o_pinpair</i>	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)
```

## **Allegro SKILL Reference**

### Logic Access Functions

---

```
pp = axlPinPair("U2.13" car(ratTs))
```

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)
```

Example 2: 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.
<i>nil</i>	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")
```

## Allegro SKILL Reference

### Logic Access Functions

---

## 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/o_dbid  
    [g_ripup]  
)  
⇒ 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

<i>t_name</i>	Net name.
<i>o_dbid</i>	<i>dbid</i> of a net.
<i>g_ripup</i>	Optional argument. Ripup associated etch when a net is deleted.

### Value Returned

<i>t</i>	Net successfully removed.
<i>nil</i>	No net is removed.

### Example

```
axlRemoveNet ("GND")
```

## 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 net names in property 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")
```

## **Allegro SKILL Reference**

### Logic Access Functions

---

```
; 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*.

## **axlSchedule**

```
axlSchedule(  
    o_net/t_net  
    [g_userSchedule]  
)  
==> t_schedule/nil
```

### **Description**

Gets net schedule. When `g_userSchedule` is `t`, this fetches the a user schedule or a partial user schedule from a net. Returns nil if the net is completely algorithm scheduled. Using `t` is recommended. When `g_userSchedule` is nil, returns the schedule of the complete net even if the net is completely algorithm scheduled.

The format of `t_schedule` is a string in the \$SCHEDULE netin (3rd party) format. See `netin` documentation for more info about the syntax.

### **Arguments**

<code>o_net</code>	<code>dbid</code> of net
<code>t_net</code>	Name of net

`g_userSchedule` (opt) If `t` returns the schedule if the net is user schedule or partial user schedule. Other nets in this netin format if nil remove

### **Values Returned**

<code>t_schedule</code>	Schedule or partial user schedule.
<code>nil</code>	Failed or net is not user schedule or partial user schedule. Use <code>axlDebug</code> to obtain more data.

### **See Also**

[axlScheduleNet](#)

## Examples

Net2 has the following pins and rat-Ts:

```
P1.7 U1.4 U1.10 U2.6 T.1
```

It is partially user schedule such as P1.7, U1.4, and U1.10 should be connected to rat-T, T.1. Pin U2.6 is algorithm scheduled but the sub-schedule (partial) indicates it should connect to U1.10 (indicated by a star '\*' in the schedule string).

Fetch partial schedule (note no U2.6)

```
q = axlSchedule("NET2" t)
=> "P1.7 T.1 U1.4 ; T.1 *U1.10 "
```

Same net but complete schedule

```
q = axlSchedule("NET2" t)
=> "P1.7 T.1 U1.4 ; T.1 *U1.10 U2.6 "
```

## axlScheduleNet

```
axlScheduleNet(  
    o_net/t_net  
    t_schedule/nil  
)  
==> t/nil
```

### Description

This applies a user schedule or a partial user schedule to a net. Format of *t\_schedule* is a string in the \$SCHEDULE netin (3rd party) format.

**Note:** See `netin` documentation for more info about the syntax.

When *t\_schedule* is *nil* it removes any user schedule or partial user schedule from the net and restores its default schedule algorithm (NET\_SCHEDULE property).

### Arguments

<i>o_net</i>	<i>dbid</i> of net
<i>t_net</i>	Name of net
<i>t_schedule</i>	Schedule or partial schedule using \$SCHEDULE netin format; if <i>nil</i> , remove

### Value Returned

<i>t</i>	Schedule applied.
<i>nil</i>	Failed or arguments are incorrect (use <code>axlDebug</code> ) for more info.

### See Also

[axlSchedule](#)

## **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 `.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

<i>o_compDefDbid</i>	Component definition of the device file to write out.
<i>t_output_dir</i>	Directory in which to write the files. If not provided, the current directory is used.

## Value Returned

<i>t</i>	Device file successfully written.
<i>nil</i>	Failed to write device file due to incorrect <i>dbid</i> 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.

## **Allegro SKILL Reference**

### Logic Access Functions

---

---

## DFM SKILL APIs

---

This chapter describes the AXL-SKILL functions related to design for manufacturing (DFM).

### Component Lead Functions

#### **axlCreateLeadContact**

```
axlCreateLeadContact (
  t_leadType
  t_symbol
  l_pins
  r_laedParams
)
⇒ t/nil
```

#### **Description**

This function creates lead contact shape for the pins belonging to a symbol definition.

#### **Arguments**

<i>t_leadType</i>	Type of leads to be created. The valid types are: "No lead", "Ball", "Ball-Collapsing", "Ball-Non collapsing", "Bump", "Column", "Pillar", "Rectangular end cap", "Cylindrical end cap", "Flat lead", "Flat lug", "Flat no-lead bottom", "Flat no-lead edge", "Flat thermal", "Gull wing", "Ribbon L-Inward", "Ribbon L-Outward", "Under body outward L", "J-Lead", "Side lead convex", "Side lead concave", "Side lead flat", "Butt lead", "Corner concave", "Other surface", "Through round", "Through rectangular", "Through other", "No connect", "Press fit"
<i>t_symbol</i>	Name of the symbol

## Allegro SKILL Reference

### DFM SKILL APIs

---

<i>l_pins</i>	List of dbids of the pins for which the contact shapes are to be created. If this is passed as <code>nil</code> all the numbered pin will be considered for lead.
<i>r_leadParams</i>	Use <code>make_axlLeadParam</code> . Elements are:  leadFigure: (text) Allowed names are "CIRCLE", "RECTANGLE", "DONUT", "N_SIDED_POLY" "OBLONG", "OCTAGON", "ROUNDED_RECT", "CHEMFERED_RECT", "SQUARE", "SHAPE"
<i>leadShape</i>	(text) name of the shape (valid only if 'SHAPE is specified as leadFigure)  figureSize: (list) (f_width:f_height) width and height of the figure  offset: (list) (x_offset:y_offset) offset of the lead contact shape with respect to pad center  length: length of the actual lead  width: width of the lead  height: height of the lead  n_sides: no of sides (valid for N_SIDED_POLY)  outer_inner_dia: (outerDia innerDia) valid for Donut shape  cornerRadOrChanf: specific to rounded or chamfered rectangle. They specify corner radius or corner chamfer.  corner: specifies the list of corners that need to be rounded/chamfered. Valid values of an element are:  "UPPER_LEFT", "UPPER_RIGHT", "LOWER_LEFT", "LOWER_RIGHT"

#### Value Returned

<code>t</code>	If successful
<code>nil</code>	If failed

#### See Also

[axlDeleteLeadContact](#)

## **Allegro SKILL Reference**

### DFM SKILL APIs

---

#### **Example**

```
leadParams = cons( make_axlLeadParam( ?leadFigure 'N_SIDED_POLY ?figureSize  
10:10 ?offset  
-1:-2 ?length 50 ?width 10 ?height 10 ?n_sides 8 ) nil )  
axlCreateLeadContact("Ball" "RES" nil leadParams) ; apply ball type lead for  
all the pins in the symbol
```

## **axlDeleteLeadContact**

```
axlDeleteLeadContact(  
    t_symbol  
    l_pins  
)  
⇒ t/nil
```

### **Description**

This function deletes lead contact shape for the pins belonging to a symbol definition.

### **Arguments**

<i>t_symbol</i>	Name of the symbol
<i>l_pins</i>	List of dbids of the pins for which the contact shapes are to be deleted. If this is passed as <code>nil</code> all the numbered pin will be considered.

### **Value Returned**

<code>t</code>	If successful
<code>nil</code>	If failed

### **See Also**

[axlCreateLeadContact](#)

### **Example**

```
axlDeleteLeadContact("RES" nil) ; delete all the lead contact for the symbol
```

## **axlGetSupportedLeadTypes**

```
axlGetSupportedLeadTypes (
    g_category
)
⇒ list of supported lead types/nil
```

### **Description**

This function returns all the supported lead types in the form of a list. Each element of the list is a name of the lead types supported by the lead editor and lead checks.

### **Arguments**

<i>g_category</i>	Category of the lead. Valid values are:
■	ALL: Get all type of leads
■	SMD: Get lead types that are valid for surface mount devices
■	THRU: Get lead types that are valid for through hole pins
■	BALL: Get lead types that are applicable to balls

### **Value Returned**

list of valid lead types	If successful
nil	If failed

### **See Also**

[axlCreateLeadContact](#), [axlDeleteLeadContact](#)

### **Example**

```
ll = axlGetSupportedLeadTypes("ALL") ; Get all supported lead types
```

## DFM CSet Operations

### Worksheet Names

The following table lists supported worksheet names.

**Table 31-1 Worksheet names**

"DFF_OUTLINE"	Indicates that cset is going to hold constraints pertaining to OUTLINE worksheet
"DFF_MASK"	Indicates that cset is going to hold constraints pertaining to MASK worksheet
"DFF_HOLE"	Indicates that cset is going to hold constraints pertaining to HOLE worksheet
"DFF_SILKSCREEN"	Indicates that cset is going to hold constraints pertaining to SILKSCREEN worksheet
"DFF_ANNULAR_RING"	Indicates that cset is going to hold constraints pertaining to ANNULAR RING worksheet
"DFF_COPPER_FEATURE"	Indicates that cset is going to hold constraints pertaining to COPPER FEATURE worksheet
"DFF_COPPER_SPACING"	Indicates that cset is going to hold constraints pertaining to COPPER SPACING worksheet
"DFA_OUTLINE"	Indicates that cset is DFA cset pertaining to DFA outline worksheet
"DFA_PKGPKG"	Indicates that cset is DFA cset pertaining to DFA package to package worksheet
"DFA_SPACING"	Indicates that cset is a DFA cset pertaining to DFA spacing worksheet
"DFA_COMPLEAD"	Indicates that cset is a DFA cset pertaining to DFA component lead worksheet
"DFA_MASK"	Indicates that cset is a DFA cset pertaining to DFA mask worksheet
"DFA_FIDUCIAL"	Indicates that cset is a DFA cset pertaining to DFA fiducial worksheet

## Allegro SKILL Reference

### DFM SKILL APIs

---

"DFT_OUTLINE"	Indicates that cset is a DFT cset pertaining to DFT outline worksheet
"DFT_MASK"	Indicates that cset is a DFT cset pertaining to DFT mask worksheet
"DFT_SPACING"	Indicates that cset is a DFT cset pertaining to DFT spacing worksheet
"DFT_PROBE"	Indicates that cset is a DFT cset pertaining to DFT probe worksheet

### **Subtype CSet Names**

The following table lists names of supported subtype CSets.

**Table 31-2 List of Subtype CSets**

"DFF_OUTLINE"	Indicates that cset is going to hold constraints pertaining to OUTLINE worksheet
"DFF_MASK"	Indicates that cset is going to hold constraints pertaining to MASK worksheet
"DFF_HOLE"	Indicates that cset is going to hold constraints pertaining to HOLE worksheet
"DFF_SILKSCREEN"	Indicates that cset is going to hold constraints pertaining to SILKSCREEN worksheet
"DFF_ANNULAR_RING"	Indicates that cset is going to hold constraints pertaining to ANNULAR RING worksheet
"DFF_COPPER_FEATURE"	Indicates that cset is going to hold constraints pertaining to COPPER FEATURE worksheet
"DFF_COPPER_SPACING"	Indicates that cset is going to hold constraints pertaining to COPPER SPACING worksheet
"DFA_OUTLINE"	Indicates that cset is DFA cset pertaining to DFA outline worksheet
"DFA_PKGPKG"	Indicates that cset is DFA cset pertaining to DFA package to package worksheet
"DFA_SPACING"	Indicates that cset is a DFA cset pertaining to DFA spacing worksheet

## **Allegro SKILL Reference**

### DFM SKILL APIs

---

"DFA_COMPLEAD"	Indicates that cset is a DFA cset pertaining to DFA component lead worksheet
"DFA_MASK"	Indicates that cset is a DFA cset pertaining to DFA mask worksheet
"DFA_FIDUCIAL"	Indicates that cset is a DFA cset pertaining to DFA fiducial worksheet
"DFT_OUTLINE"	Indicates that cset is a DFT cset pertaining to DFT outline worksheet
"DFT_MASK"	Indicates that cset is a DFT cset pertaining to DFT mask worksheet
"DFT_SPACING"	Indicates that cset is a DFT cset pertaining to DFT spacing worksheet
"DFT_PROBE"	Indicates that cset is a DFT cset pertaining to DFT probe worksheet

## **axlCreateDfmAssignment**

```
axlCreateDfmAssignment(  
    className  
    worksheetName  
    dfmentryType  
    stackupName  
    subclassGrouporGLTName  
    cset_name  
    subClassName  
)  
⇒ t/nil
```

### **Description**

This function assigns a DFM CSet to a layer, layer group, or a subclass group. With this all the constraints associated with the CSet gets assigned to the object. The DRC engine uses the CSet assignments with the objects to determine the constraints that needs to be evaluated.

## Allegro SKILL Reference

### DFM SKILL APIs

---

#### Arguments

<i>className</i>	Name of the classes on which the constraint is applicable. This is applicable only form non etch type of rules specified through subclass group.
<i>worksheetName</i>	Name of the worksheet where the rule is applicable. Refer to <a href="#">Worksheet names</a> for valid worksheet names
<i>dfmentryType</i>	<p>Specifies the entry type in the assignment table. The valid values are:</p> <p>'LAYER_BASED': Indicates that assignment is layer based and is assigned to a layer.</p> <p>'CLASS_SUBCLASS': Indicates that the assignment is class subclass based. It needs the subclass group name to be filled.</p> <p>'GLT_BASED': Indicates that the assignment is applicable to a generic layer type. The subclass group name argument should have a valid GLT name if this type is selected.</p> <p>Valid GLT names are:</p> <p>"TESTING LAYERS": for DFT</p> <p>"ASSEMBLY LAYERS": for DFA</p> <p>"CONDUCTOR": For DFF</p> <p>"PLANE"</p> <p>Any GLT name defined in cross-section editor</p> <p>'STACKUP_BASED': Indicates that the rule is assigned to the stackup. That means that the rule will be assigned to the entire stackup (For example, Outline to Cutout rule).</p>
<i>stackupName</i>	Specifies the name of the stackup on which the rule is applicable. The primary stackup is named as "PRIMARY"
<i>subclassGrouporGLTName</i>	Specifies that the name field in the table corresponds to a bucket name which is essentially a group of class and subclass combination. This interpretation is valid only if the entry type is CLASS_SUBCLASS. If value of entry type is GLT_BASED, this field refers to GLT name.

<i>cset_name</i>	Specifies the name of the DFF CSet that corresponds to the mapping based on other fields.
<i>subClassName</i>	Specifies the subclass name. This holds the name of the layer or subclass in the table.

### Value Returned

<i>t</i>	If successful
<i>nil</i>	If failed

### See Also

[axlCreateDffCset](#), [axlDeleteDffCset](#)

### Example

```
ll = axlCreateDFMCset("test_cset33" "dff_outline" "ETCH")
axlCreateDfmAssignment( "ETCH" "dff_outline" "layer_based" "primary" "dummy"
"test_cset33" "TOP")
```

## **axlDFMIsCsetLocked**

```
axlDFMIsCsetLocked(  
    worksheetName  
    cset_name  
)  
⇒ t/nil
```

### **Description**

This function Checks if a DFM cset is locked or not.

### **Arguments**

<i>worksheet_name</i>	Name of the worksheet where the rule is applicable. Refer to <a href="#">Worksheet names</a> for valid worksheet names
<i>cset_name</i>	Specifies the name of the DFM CSet that corresponds to the mapping based on other fields.

### **Value Returned**

t	If locked
nil	If not locked

### **See Also**

[axlDFMCsetLock](#)

### **Example**

```
lockStatus = axlDFMIsCsetLocked( "dff_outline" "test_cset33" )
```

## **axIDFMCsetLock**

```
axIDFMCsetLock(  
    worksheetName  
    cset_name  
    g_mode  
)  
⇒ t/nil
```

### **Description**

This function locks/unlocks a DFM cset.

### **Arguments**

<i>worksheet_name</i>	Name of the worksheet where the rule is applicable. Refer to <a href="#">Worksheet names</a> for valid worksheet names
<i>cset_name</i>	Specifies the name of the DFM cset that corresponds to the mapping based on other fields.
<i>g_mode</i>	May either be <code>t</code> (to lock) or <code>nil</code> to unlock

### **Value Returned**

<code>t</code>	If updated lock status
<code>nil</code>	If an error

### **See Also**

[axIDFMIIsCsetLocked](#)

### **Example**

```
axIDFMCsetLock( "dff_outline"  "test_cset33"  t)
```

## **axlDeleteDfmAssignment**

```
axlDeleteDfmAssignment(  
    worksheetName  
    cset_name  
)  
⇒ t/nil
```

### **Description**

This function deletes Design For manufacturing cset assignment.

### **Arguments**

<i>worksheet_name</i>	Name of the worksheet where the rule is applicable. Refer to <a href="#">Worksheet names</a> for valid worksheet names
<i>cset_name</i>	Specifies the name of the DFM cset that corresponds to the mapping based on other fields.

### **Value Returned**

t	If successful
nil	If failed

### **See Also**

[axlCreateDFMCset](#)

### **Example**

```
axlDeleteDfmAssignment( "dff_outline" "test_cset33" )
```

## **axlGetDFMAssignedCsets**

```
axlGetDFMAssignedCsets(  
    subtype  
)  
⇒ l_dff_cset/nil
```

### **Description**

This function gets the Assigned Design For Manufacturing (DFM) CSets in a design.

## Arguments

*subtype* Specifies the sub type of the CSet. The valid values are:

DFF\_OUTLINE: Indicates that CSet is going to hold constraints pertaining to OUTLINE worksheet.

DFF\_MASK: Indicates that CSet is going to hold constraints pertaining to MASK worksheet.

DFF\_HOLE: Indicates that CSet is going to hold constraints pertaining to HOLE worksheet.

DFF\_SILKSCREEN: Indicates that CSet is going to hold constraints pertaining to SILKSCREEN worksheet.

DFF\_ANNULAR\_RING: Indicates that CSet is going to hold constraints pertaining to ANNULAR RING worksheet.

DFF\_COPPER\_FEATURE: Indicates that CSet is going to hold constraints pertaining to COPPER FEATURE worksheet.

DFF\_COPPER\_SPACING: Indicates that CSet is going to hold constraints pertaining to COPPER SPACING worksheet.

DFA\_OUTLINE: Indicates that CSet is a DFA CSet pertaining to DFA outline worksheet.

DFA\_PKGPKG: Indicates that CSet is a DFA CSet pertaining to DFA package to package worksheet.

DFA\_SPACING: Indicates that CSet is a DFA CSet pertaining to DFA spacing worksheet.

DFA\_COMPLEAD: Indicates that CSet is a DFA CSet pertaining to DFA component lead worksheet.

DFA\_MASK: Indicates that CSet is a DFA CSet pertaining to DFA mask worksheet.

DFA\_FIDUCIAL: Indicates that CSet is a DFA CSet pertaining to DFA fiducial worksheet.

DFT\_OUTLINE: Indicates that CSet is a DFT CSet pertaining to DFT outline worksheet.

DFT\_MASK: Indicates that CSet is a DFT CSet pertaining to DFT mask worksheet.

DFT\_SPACING: Indicates that CSet is a DFT CSet pertaining to DFT spacing worksheet.

DFT\_PROBE: Indicates that CSet is a DFT CSet pertaining to DFT probe worksheet.

### **Value Returned**

List of CSets and types along with the assignment details, if there is at least one assigned CSet. If there are more than one CSet on that category, it will be a list of lists.

Each list will contain the following:

1. Name of the cset
2. Subtype of the cset. The valid values are mentioned under arguments.
3. DFM entry type. The valid values are as follows:
  - ❑ LAYER\_BASED: Indicates that assignment is layer based and is assigned to a layer.
  - ❑ CLASS\_SUBCLASS: Indicates that the assignment is class and subclass based. It needs the subclass group name to be filled.
  - ❑ GLT\_BASED: Indicates that the assignment is applied to a generic layer type. Valid GLT names are:
    - TESTING LAYERS: for DFT
    - ASSEMBLY LAYERS: for DFA
    - CONDUCTOR: For DFF
    - PLANE
    - Any GLT name defined in Cross-section Editor.
  - ❑ STACKUP\_BASED: Indicates that the rule is assigned to the stackup. For example,, Outline to Cutout rule.
4. stackupName: Specifies the name of the stackup on which the rule is applied. The primary stackup is named as PRIMARY.

5. subclassGrouporGLTName: Specifies that the name field in the table corresponds to a subclass group name which is essentially a group of class subclass combination. This interpretation is valid only if the entry type is CLASS\_SUBCLASS. If the value of entry type is GLT\_BASED, this field refers to GLT name. Otherwise subclass name is returned.
6. subclass\_name: Specifies the subclass name. This field holds the name of the layer or subclass in the table.
7. className: Specifies the name of the class on which the constraint is applicable. This field is allowed only for non-etch type of rules specified through subclass group. By default, it returns BOARD\_GEOMETRY.

nil is returned if failed or if there is no CSet assigned.

## See Also

[axlCreateDFMCset](#), [axlCreateDfmAssignment](#), [axlDeleteDffCset](#)

## Example

For example, in a design that has 2 CSets with types Annular ring and 1 CSet with type Mask in DFF sub-domain:

- The function call axlGetDFMAssignedCsets("DFF\_ANNULAR\_RING") return the following:

```
( ("ABC1" "DFF_ANNULAR_RING" "LAYER_BASED" "PRIMARY" "SOLMPH_TOP"
  "SOLMPH_TOP" "BOARD_GEOMETRY") ("TEST2" "DFF_ANNULAR_RING" "LAYER_BASED"
  "PRIMARY" "SOLMPH_TOP" "SOLMPH_BOT" "BOARD_GEOMETRY") )
```
- Function call axlGetDFMAssignedCsets("DFF\_MASK") return the following:

```
("ABC2" "DFF_MASK" "CLASS_SUBCLASS" "PRIMARY" "SILK2PAD_TOP" "PLATING_BAR"
  "BOARD_GEOMETRY") )
```

## **axlGetDFMCsetAssignmentStatus**

```
axlGetDFMCsetAssignmentStatus (
    cset_name
    subtype
    => l_dff_cset/nil
```

### **Description**

This function checks if a given Manufacturing (DFM) CSet is assigned in a design.

## Arguments

*cset\_na* Name of the CSet

*me*

*subtype* Specifies the sub type of the CSet. The valid values are:

DFF\_OUTLINE: Indicates that CSet is going to hold constraints pertaining to OUTLINE worksheet.

DFF\_MASK: Indicates that CSet is going to hold constraints pertaining to MASK worksheet.

DFF\_HOLE: Indicates that CSet is going to hold constraints pertaining to HOLE worksheet.

DFF\_SILKSCREEN: Indicates that CSet is going to hold constraints pertaining to SILKSCREEN worksheet.

DFF\_ANNULAR\_RING: Indicates that CSet is going to hold constraints pertaining to ANNULAR RING worksheet.

DFF\_COPPER\_FEATURE: Indicates that CSet is going to hold constraints pertaining to COPPER FEATURE worksheet.

DFF\_COPPER\_SPACING: Indicates that CSet is going to hold constraints pertaining to COPPER SPACING worksheet.

DFA\_OUTLINE: Indicates that CSet is a DFA CSet pertaining to DFA outline worksheet.

DFA\_PKGPKG: Indicates that CSet is a DFA CSet pertaining to DFA package to package worksheet.

DFA\_SPACING: Indicates that CSet is a DFA CSet pertaining to DFA spacing worksheet.

DFA\_COMPLEAD: Indicates that CSet is a DFA CSet pertaining to DFA component lead worksheet.

DFA\_MASK: Indicates that CSet is a DFA CSet pertaining to DFA mask worksheet.

DFA\_FIDUCIAL: Indicates that CSet is a DFA CSet pertaining to DFA fiducial worksheet.

## **Allegro SKILL Reference**

### DFM SKILL APIs

---

DFT\_OUTLINE: Indicates that CSet is a DFT CSet pertaining to DFT outline worksheet.

DFT\_MASK: Indicates that CSet is a DFT CSet pertaining to DFT mask worksheet.

DFT\_SPACING: Indicates that CSet is a DFT CSet pertaining to DFT spacing worksheet.

DFT\_PROBE: Indicates that CSet is a DFT CSet pertaining to DFT probe worksheet.

#### **Value Returned**

t	If CSet is assigned
nil	Otherwise

#### **See Also**

[axlCreateDFMCset](#), [axlCreateDfmAssignment](#), [axlDeleteDffCset](#),  
[axlGetDFMAssignedCsets](#)

#### **Example**

```
RetVal = axlGetDFMCsetAssignmentStatus( "ABC" "DFF_MASK")
```

## axlRenameDFFCset

```
axlRenameDFFCset (
    oldCsetName
    newCsetName
    csetType
)
⇒ l_csetObj/nil
```

### Description

This function renames an existing DFF CSet with a new name.

### Arguments

<i>oldCsetName</i>	Name of the existing cset
<i>newCsetName</i>	New name of the cset
<i>csetType</i>	Type of the existing cset. Refer to <a href="#">List of Subtype CSets</a> for valid values.

### Value Returned

<i>l_csetObj</i>	New id of cset
<i>nil</i>	If failed

### See Also

[axlCreateDffCset](#), [axlCreateDffAttr](#)

### Example

```
axlRenameDFFCset("test" "test_new" "dff_outline")
==> "12,1,2,0,test_bucket1,Top,test_cset1"
```

## **axlCreateDFMCset**

```
axlCreateDFMCset(  
    name  
    subtype  
    usage  
)  
⇒ l_dff_set/nil
```

### **Description**

This function creates a Design For Manufacturing (DFF) Cset. The DFM cset holds the constraint values for various checks to be performed on the design.

### **Arguments**

<i>name</i>	Name of the cset to create
<i>subtype</i>	Sub type of the cset.  Refer to <a href="#">List of Subtype CSets</a> for valid values.
<i>usage</i>	Specifies the cset intent. The valid values are:  "ETCH": Indicates that cset is to be used for constraints in conducting layer. This is valid for DFF, DFA and DFT subdomain  "NONETCH": Indicates that the cset is for constraints on non-etch subclasses. Valid for all the DFM subdomains  "STACKUP": Indicates that the cset captures constraints for stackup based rules. Valid for DFF  "SPACING": Indicates that cset captures package to package rules. This is valid for DFA_PKGPKG worksheets.  "HEIGHT": Indicates that cset captures constraints for AOI rules. This is valid for DFA_PKGPKG worksheet.

### **Value Returned**

l_dfm_cset	db id of the DFM cset object, if successful
nil	If failed

**See Also**

`axlDeleteDffCset`

**Example**

```
axlCreateDFMCset("test_cset33" "OUTLINE")
```

## **axlDeleteDFMCset**

```
axlDeleteDffCset(  
    name  
    subtype  
)  
⇒ t/nil
```

### **Description**

This function deletes a DFM CSet from the database.

### **Arguments**

<i>name</i>	Name of cset to delete
<i>subtype</i>	Type of the DFM cset. Refer to <a href="#">List of Subtype CSets</a> for valid values.

### **Value Returned**

t	If successful
nil	If failed

### **See Also**

[axlCreateDFMCset](#)

### **Example**

```
axlCreateDFMCset("test_cset33" "dff_outline")
```

## **axlGetDFMCsets**

```
axlGetDFMCsets(  
    subtype  
)  
⇒ l_dff_cset/nil
```

### **Description**

This function gets the Design For Manufacturing (DFM) Csets in the design.

### **Arguments**

*subtype* Sub type of the cset.  
Refer to [List of Subtype CSets](#) for valid values.

### **Value Returned**

l_dfm_csetlist	List of DB ids for the DFM csets of the specified types
nil	Failed or if there is none

### **See Also**

[axlCreateDFMCset](#), [axlDeleteDffCset](#)

### **Example**

```
axlCreateDFMCset("DFT_PROBE")
```

## **axlGetDFMCset**

```
axlGetDFMCset(  
    name  
    subtype  
)  
⇒ l_dff_cset/nil
```

### **Description**

This function gets the Design For Manufacturing (DFM) Csets in the design.

### **Arguments**

<i>name</i>	Name of the cset
<i>subtype</i>	Sub type of the cset. Refer to <a href="#">List of Subtype CSets</a> for valid values.

### **Value Returned**

l_dff_cset	dbid of the cset
nil	Failed or if the cset does not exist

### **See Also**

[axlCreateDFMCset](#), [axlGetDFMCsets](#)

### **Example**

```
axlGetDFMCset("Test_cset" "DFT_PROBE" )
```

## **Allegro SKILL Reference**

### **DFM SKILL APIs**

---