cadence™

Allegro® User Guide: SKILL Reference

Product Version 16.3
December 2009

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

Contents

10
Form Interface Functions

13
Design Control Functions 683

Alphabetical List of Functions

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 manufacturability.

If you are reading this manual for a general understanding of AXL capabilities, but do not actually intend to program in AXL, read Chapter 1, “Introduction to 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.

italics	Data type name.
nil	Standard SKILL for empty list; as a return value, may indicate failure.
t	Standard SKILL for “true;” return value for success.
[name]	Optional argument “name.”
<name>	Argument of type “name” required.
dbid	Instance of an Allegro PCB Editor database object (means “database id”)
figure	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.

l_bBox	A list of the coordinates of a bounding box. Coordinate pairs are lower left and upper right. For example, `(list( 100:100 200:200 ))`
t_layer	A pair of names separated by a slash "/" denoting the name of an Allegro PCB Editor class-subclass. For example, "PACKAGE GEOMETRY/SILKSCREEN_TOP."
lo_dbid	Function that takes or returns a dbid or list of dbids.
o_dbid	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 P-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 `axlName` 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: ■ AXL functions ■ AXL initialization, environment ■ Starting and stopping AXL ■ Debugging AXL programs ■ *dbids* and object persistence ■ Selecting Allegro PCB Editor database objects ■ AXL–SKILL database object types	Chapter 1, “Introduction to Allegro PCB Editor SKILL Functions”
The structure of Allegro PCB Editor AXL database objects, and how they are related to each other. Lists attributes of each object type. ■ Database rules ■ Attribute types ■ Figure (geometric) database types ■ Logical database types ■ Property database types ■ Full attribute listing for all Allegro PCB Editor database objects	Chapter 2, “The Allegro PCB Editor Database User Model”
Path structures and AXL functions that add path objects and other figure objects to the Allegro PCB Editor database. ■ Path create functions ■ Create shape and rectangle functions ■ Create functions for line, pin, symbol, text, and via	Chapter 14, “Database Create Functions”

For Information About . . .	Read . . .
Read/Write access to Allegro PCB Editor database parameter objects.	Chapter 3, “Parameter Management Functions”
The select set and AXL functions for managing the select set and selecting single and multiple database objects. ■ Point selection ■ Box selection ■ Selection by name and object *dbid* ■ Find filter management ■ Selection set management	Chapter 4, “Selection and Find Functions”
AXL functions that operate on database objects in the same way as interactive Allegro PCB Editor commands, including functions to: ■ Delete objects ■ Show objects	Chapter 5, “Interactive Edit Functions”
Reading database objects: ■ Opening an Allegro PCB Editor design ■ Accessing standalone branch figures, properties, pads, and text	Chapter 6, “Database Read Functions”
AXL functions for ■ 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	Chapter 7, “Allegro PCB Editor Interface Functions”

For Information About . . .	Read . . .
AXL functions for ■ Setting Allegro PCB Editor shell variables ■ Sending a command string to the Allegro PCB Editor shell.	Chapter 8, “Allegro PCB Editor Command Shell Functions”
AXL functions for ■ Prompting and getting confirmation from the user, displaying text files ■ Displaying and printing ASCII files	Chapter 9, “User Interface Functions”
AXL forms and functions for ■ Creating forms, including the various types of form fields ■ Setting up callbacks for response to input to individual fields	Chapter 10, “Form Interface Functions”
AXL functions related to Simple Graphics Drawing	Chapter 11, “Simple Graphics Drawing Functions”
Writing AXL functions for ■ Setting up for user messages ■ Displaying messages to users	Chapter 12, “Message Handler Functions”

For Information About . . .	Read . . .
AXL functions for ■ 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	Chapter 13, “Design Control Functions”

For Information About . . .	Read . . .
AXL functions for ■ Creating and removing database groups. ■ Adding and removing database objects from database groups.	Chapter 15, “Database Group Functions”
AXL functions for ■ Creating, changing, and deleting database attachments ■ Checking whether an object is a database attachment ■ Getting the ids of all database attachments ■ Getting a database attachment	Chapter 16, “Database Attachment Functions”
AXL functions for ■ Improving the performance and program memory use while updating many etch or package symbols in batch mode ■ Marking the start of a database transaction and returning the mark to the calling function ■ Writing a mark in the database to allow future rollback or commitment ■ Commiting and undoing a database transaction	Chapter 17, “Database Transaction Functions”

For Information About . . .	Read . . .
AXL functions for ■ Getting and setting current DRC modes and values for design constraints and ECset members ■ Creating, deleting, and getting the dbid of an ECset ■ Checking the syntax of a given value against the allowed syntax for a given constraint ■ Batching and tuning DRC updates from constraint changes made using axlCNS<xxx> functions	Chapter 18, “Constraint Management Functions”
AXL functions for ■ Registering and unregistering SKILL commands with the command interpreter ■ Getting and setting the controls for line lock, active layer ■ Defining popups ■ Getting user data	Chapter 19, “Command Control Functions”
Polygon operation functions, attributes, and use models.	Chapter 20, “Polygon Operation Functions”
Getting Allegro PCB Editor file names, and opening and closing files	Chapter 21, “Allegro PCB Editor File Access Functions”
AXL functions for ■ SKILL access to the extract command ■ Selecting sets of database objects as members of a view and applying an AXL function to each	Chapter 22, “AXL-SKILL Data Extract Functions”

For Information About . . .	Read . . .
AXL functions for ■ Calculating an arc center given various data ■ Converting quantities to various units	Chapter 23, “Utility Functions”
Using math utility functions.	Chapter 24, “Math Utility Functions”
Odds and ends.	Chapter 25, “Database Miscellaneous Functions”
Using logic access functions.	Chapter 28, “Logic Access Functions”

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.	*Allegro PCB Editor User Guide: Getting Started with Physical Design*
Building and managing libraries, including defining padstacks, custom pads, packages, electrical attributes, and formats.	*Allegro PCB Editor User Guide: Defining and Developing Libraries*

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

Customer Support

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

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

SourceLink

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

http://sourcelink.cadence.com

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

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:

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

User Discussion Forums

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

http://www.cadenceusers.org

1

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

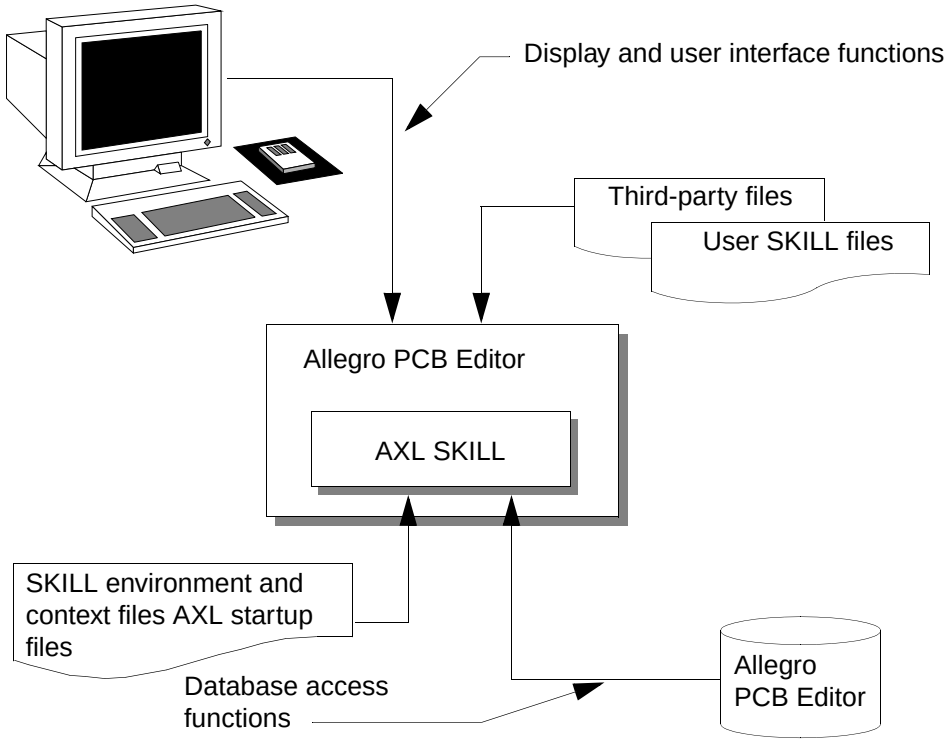
Later chapters describe in detail the AXL database objects and all AXL-SKILL functions.

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

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

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 dnotes 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 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: `arc segment`).
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: `line segment`).
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: `lines and clines`).
PPin	Is a physical instance of a pin with associated padstack.
Polygon	Is an unfilled, closed path (Allegro PCB Editor figure: `unfilled shape`).
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.

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
```

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
- 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

Object Name	Object Parts
Property Dictionary	Property Entries
Net	Branches
Padstack	Pads

Table 1-1 Other Database Object Relationships

Object Name	Object Parts
Polygon	Paths
Shape	Paths
Void	Polygons
Polygon	Paths
PPin	Pin Number Text
Layer Group	Layers
Textblock Group	Textblocks

2

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 76)
  - Branches (Table 2-4 on page 76)
  - Design Files (Table 2-5 on page 77)
  - Drcs (Table 2-6 on page 77)
  - Lines (Table 2-9 on page 79)
  - Paths (Table 2-12 on page 81)
  - Polygons (Table 2-14 on page 83)
  - Ppins (Table 2-13 on page 81)
  - Shapes (Table 2-16 on page 84)
  - Symbols (Table 2-17 on page 85)
  - Tees (Table 2-19 on page 86)
  - Vias (Table 2-21 on page 87)
  - Pads (Table 2-10 on page 79)
  - Padstacks (Table 2-11 on page 80)

- Symdefs (Table 2-18 on page 86)

- Logical objects
  - Components (Table 2-24 on page 89)
  - Functions (Table 2-26 on page 90)
  - Function Pins (Table 2-27 on page 91)
  - Nets (Table 2-29 on page 93)
- Property dictionary objects (Table 2-33 on page 97)
- Parameter objects
  - Design (Table 2-36 on page 100)
  - Display (Table 2-37 on page 101)
  - Layer Group (Table 2-39 on page 102)
  - Layer (Table 2-40 on page 103)
  - Textblock Group (Table 2-41 on page 104)
  - Textblock (Table 2-42 on page 104)

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 72 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.

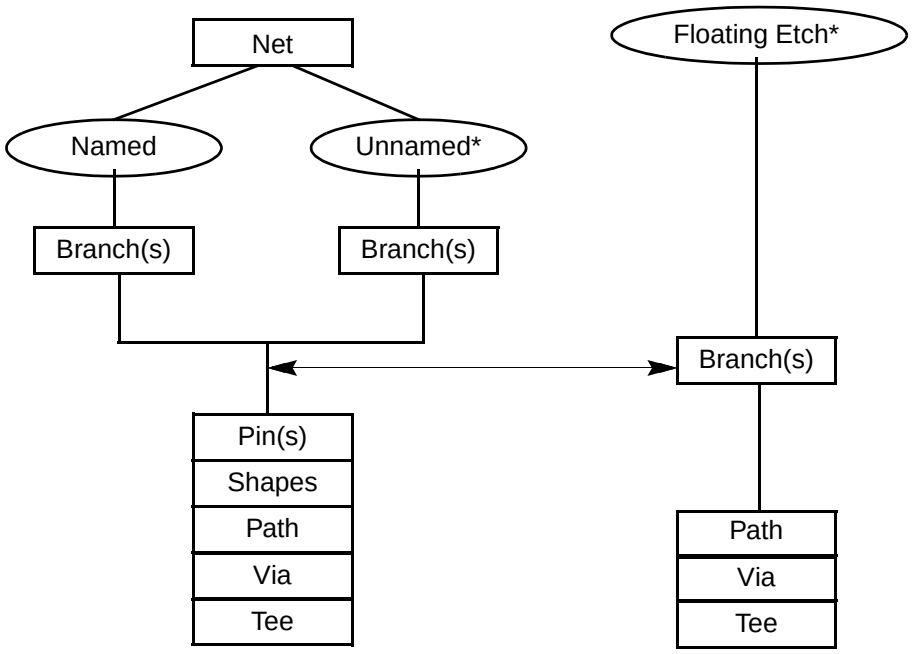
- Deleting a segment from a path so it detaches from a pin, via, tee, or shape causes the etch to be classed as *floating*.

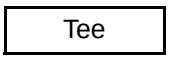
That means the etch is not a member of any net. A floating etch has a `nil` netname.

- 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.

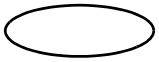
Figure 2-1 Allegro PCB Editor Connectivity Model



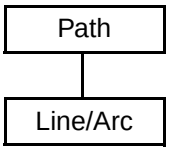


Connection of three or more paths, added and removed as needed

Can be deleted by ripup or glossing operations



The net attribute on paths, vias and tees changes due to connectivity to pins and shapes.



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 *`dbids`*
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` = cannot modify object with AXL function

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

Figure Database Types

Figures, in 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 75 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 75) and the Generic Allegro PCB Editor Object attributes (see Table 2-1 on page 74) also apply to these figure types.

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

Table 2-3 Arc Attributes

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

Table 2-4 Branch Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
children	l_dbid	List of *`dbids`* 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`

Table 2-5 Design Attributes

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

Table 2-6 DRC Attributes

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

Table 2-6 DRC Attributes, *continued*

Attribute Name	Type	Description
expected	string	Expected value (user units)
fixed	t/nil	`t` = Allegro PCB Editor generated DRC `nil` = user defined DRC
objType	string	Type of object, in this case `"drc"`
parent	dbid	Design *`dbid`*
source	string	DRC source (property or constraint set name)
type	string	DRC type
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 Group Attributes

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

Table 2-8 Module Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
bBox	bBox	Bounding box of all physical members of group

Table 2-8 Module Attributes

Attribute Name	Type	Description
groupMembers	l_dbid	List of members of the module
name	string	Name of the module
objType	string	Type of object, in this case `"group"`
type	string	“`MODULE`”

Table 2-9 Line Attributes

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

Table 2-10 Pad Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bBox	bBox	Bounding box. Coordinates are always relative.
figure	lr_path	List of *`r_paths`* defining pad’s boundary *`lr_path`* always contains at most one *`r_path`* `nil` denotes a null pad
figureName	string	Name of pad figure is one of the following: `CIRCLE`, `SQUARE OBLONG`, `RECTANGLE`, `SHAPE` or `nil` (if drill only)

Table 2-10 Pad Attributes

Attribute Name	Type	Description
flash	string	If of type `FLASH`, name of flash symbol otherwise an empty string ""
		(Obsolete; use name attribute)
layer	string	Pad layer
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 *`dbid`*
readOnly	t	Cannot be modified.
type	string	Pad type is one of: `REGULAR`, `ANTI`, or `THERMAL`

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

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

Table 2-12 Path Attributes

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

Table 2-13 Pin Attributes

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

Table 2-13 Pin Attributes, *continued*

Attribute Name	Type	Description
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.[1] To access a particular pad, use `axlDBGetPad`.
parent	dbid	*`dbid`* of symbol owning this pin[2] `nil` if pin is standalone (as it is in a symbol drawing)
relRotation	float	Pin rotation (relative to symbol)
relxy	point	Location (relative to symbol)
rotation	float	Pin rotation (absolute)
startEnd	lt_layer	Range of layers spanned by pin[2]
testPoint	t_layer/nil	*`t_layer`*, denotes layer of testpoint `nil` = pin is not a testpoint
use	string	Pin use as shown by show element
xy	point	Location of pin in absolute coordinates

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

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

Table 2-14 Polygon Attributes

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

Table 2-15 Rat_T Attributes

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

Table 2-16 Shape Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
bBox	bBox	Bounding box
branch	dbid/nil	Branch owner
children	l_dbid	Used when a Boundary Shape points to a list of dynamic shapes
connect	l_dbid/nil	List of connected figures
fill	g_fill/t/nil	Fill pattern (see `axlDBCreateOpenShape` on page 762). `t` = filled; `nil` = unfilled Each `axlFillType` has spacing width, origin, and angle.
fillOOD	t/nil	Dynamic fill is out of date. `t` = shape needs fill updating (Only dynamic shapes can be `t`) `nil` = shape does not refill
holes	list	List of o_polygons
isHole	t/nil	`t` = polygon is a hole `nil` = polygon is not a hole
isRect	t/nil	`t` = shape is a rectangle
nSegs	integer	Number of segments in polygon
objType	string	Type of object, in this case `"shape"`
parent	dbid	`dbid` `nil` = no parent
priority	integer/nil	If shape is a dynamic shape boundary this is an integer voiding priority. For all other shapes this is `nil`. The priority is relative to other dynamic shapes on the same layer. If two dynamic shapes are coincident, the shape with the higher priority wins in voiding. This number is re-calculated as needed, so use only for comparison purposes.
segments	l_dbid	Path boundary of shape

Table 2-16 Shape Attributes, *continued*

Attribute Name	Type	Description
shapeAuto	l_dbid/nil	If dynamic shape list of generated shapes on the matching auto-gen ETCH layer
shapeBoundary	dbid/nil	If this shape is generated from a dynamic shape, this points to that shape.
shapeIsBoundary	t/nil	This shape is a dynamic shape, for example, on BOUNDARY class.
symbolEtch	dbid	Symbol owner, if etch
voids	l_dbid	List of polygon boundaries defining voids in this shape

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

Table 2-17 Symbol Attributes

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

Table 2-17 Symbol Attributes, *continued*

Attribute Name	Type	Description
type	string	Symbol type is one of: `PACKAGE`, `MECHANICAL`, `FORMAT` or `SHAPE`
xy	point	Symbol location

Table 2-18 Symdef (Symbol Definition) Attributes

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

Table 2-19 Tee Attributes

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

Table 2-20 Text Attributes

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

Table 2-21 Via Attributes

Attribute Name	Type	Description
Also includes generic object and figure attributes		
branch	dbid/nil	Branch owner
definition	dbid	Padstack definition
isMirrored	t/nil	`t` = via mirrored
isThrough	t/nil	`t` = through via
mirrorType	string	Type of mirror.
name	string	Padstack name
objType	string	Type of object, in this case `"via"`
pads	l_dbid	Unordered list of pads. To access a specific pad, use `axlDBGetPad`.
parent	dbid	Symdef or `nil`
rotation	float	Via rotation
startEnd	lt_layer	Start and end layer of via

Table 2-21 Via Attributes, *continued*

Attribute Name	Type	Description
testPoint	t_layer/nil	Via test point state is one of: ETCH/TOP or ETCH/BOTTOM
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 70): 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 2-22 Bus Attributes

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

Table 2-23 Compdef Attributes

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

Table 2-24 Component Attributes

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

Table 2-24 Component Attributes, *continued*

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

Table 2-25 Diffpair Attributes

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

Table 2-26 Function Attributes

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

Table 2-26 Function Attributes

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

Table 2-27 Function Pin Attributes

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

Table 2-28 MATCH_GROUP Attributes

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

Note: For more information, see `axlMatchGroupCreate`.

Table 2-29 Net Attributes

Note:

Attribute Name	Type	Description
Also includes generic object attributes		
branches	l_dbid	List of branches
name	string	Net name
bus	dbid	Bus dbid if part of a bus
nBranches	integer	Number of branches (when exactly one, net is fully connected) **Note:** 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`) **Note:** If a net is a member of an xnet, all pinpairs appear on the xnet.
ratsnest	l_dbid	List of ratsnest for net.
ratT	l_dbid	List of rat_T's. If none exist, this is NULL.
rpd	l_rpd	List of lists for each member net of a match group (`mg_dbid t_relatePropDelay`). For more information, see `axlMatchGroupCreate`
scheduleLocked	t/nil	`t` = net schedule cannot be changed
unconnected	integer	Number of remaining connections. This does not include connections to unplaced symbols.
unplaced	integer	Number of unplaced pins.

Table 2-30 Pinpair Attributes

Attribute Name	Type	Description
Also includes generic object attributes		

Table 2-30 Pinpair Attributes, *continued*

Attribute Name	Type	Description
ECset derived	t/nil	If `t`, pinpair was created from an ECset. `Nil` indicates pinpair created due to net override property.
groupMembers	l_dbid	List of two pins making up pinpair.
name	string	Name of the pinpair (`<refdes>. <pin#>: <refdes>. <pin#>`)
objType	string	"`group`"
parent	l_dbid	Net or xnet owning the pinpair.
parentGroups	l_dbid	Lists match groups that have this pinpair. May also list other parent groups.
rpd	l_rpd	For each match group that has this pinpair as a member, will list as a list of lists. (`mg_dbid t_relatePropDelay`)
type	string	"`PIN_PAIR`"

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

Table 2-31 RATSNEST Attributes

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

Table 2-32 Xnet Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
bus	dbid	Bus dbid if part of a bus.
diffpair	dbid	Diffpair dbid if part of a diffpair.
groupMembers	l_dbid	List of nets of the xnet.
name	string	Name of the xnet.
objType	string	"`group`"
pinpair	l_dbid	List of pinpairs associated with xnet.
rpd	l_rpd	For each match group tha has this xnet as a member, lists as a list of lists (`mg_dbid t_relatePropDelay`).
type	string	"`XNET`".

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

Property Dictionary Database Types

The following section contains tables listing the attributes of 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 2-33 Property Dictionary Attributes

Attribute Name	Type	Description
Also includes generic object attributes		
dataType	string	Data type for property value (see Allowed Property Data Types on page 96)
name	string	Name of property
objType	string	Type of object, in this case `"propDict"`
range	lf_range	Optional limits for value
units	string	Optional value units
useCount	integer	Number of objects using property
write	t/nil	`t` = writable or user defined `nil` = read-only or 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 `axlParamSet`, respectively. See Chapter 3, "Parameter Management Functions" for functions that provide easier access to layers and other parameter objects.

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

Table 2-34 Artwork Parameter Attributes

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

Table 2-35 Artwork Film Parameter Attributes

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

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

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

Table 2-36 Design Parameter Attributes

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

Notes:

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

  Setting drawing accuracy to a value greater than 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.

Table 2-37 Display Parameter Attributes

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

Table 2-38 ECset Parameter Attributes

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

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

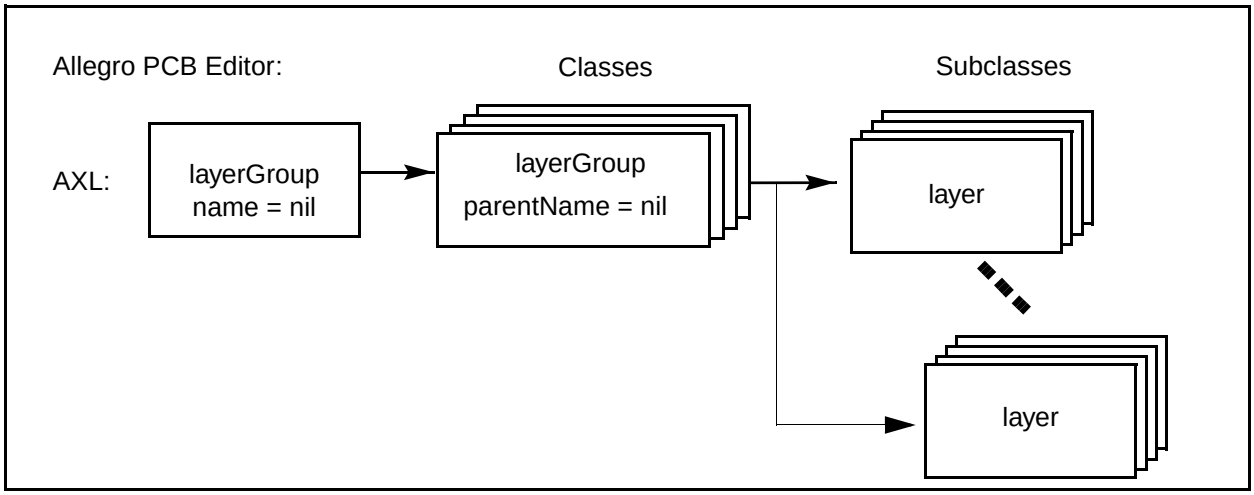


Table 2-39 Layer Group Parameter Attributes (Allegro Classes)

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
color	yes	integer	Layer color index; -1 if all child layers are not the same color
groupMember	no	l_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"`
parentName	no	string	Name of parent (`nil` if at class level)
visible	yes	t/nil	All subclasses of this class are visible

Table 2-40 Layer Parameter Attributes (Allegro Subclasses)

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
color	yes	integer	Subclass color number
drcPhotoType	no	l_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
thickness	no	string	Layer thickness (etch only)
type	no	string	"Postive" or Negative (etch only)
userDefined	No	t/nil	Is layer user defined. User defined layers can 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. Allegro PCB Editor's define etch form does not match this model. AXL does not support access to the dielectric pseudo-layers nor does it support analysis layer attributes

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

Table 2-41 Textblock Group Parameter Attributes

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

Table 2-42 Textblock Parameter Attributes

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

Table 2-43 Testprep Parameter Attributes

Attribute Name	Set?	Type	Description
Includes no generic object attributes			
allowUnderComp	yes	t/nil	Allows test pads under components.
autoInsert	yes	t/nil	Allows automatic generation of test points as needed.

Table 2-43 Testprep Parameter Attributes, *continued*

Attribute Name	Set?	Type	Description
bareBoard	yes	t/nil	If `nil`, you can only test component pins on noncomponent design side. If `t`, then you can check pins on either side of the design, as long as padstack is defined on that side.
directTest	yes	t/nil	Allows pins to be selected as test point.
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 *`testPadType`* attribute. If relative name is given, uses database then `PSMPATH` to find pad.

Table 2-43 Testprep Parameter Attributes, *continued*

Attribute Name	Set?	Type	Description
testPadDB	no	dbid	*dbid* of the testPad
testPadType	yes	symbol	Type of padstacks for probes: `smd`, `through`, or `either`.
testVia	yes	nil/string	Padstack that you use for testpads on the probe side of the design. Must be through hole. If relative name is given, uses database then `PSMPATH` to find pad.
testViaDB	no	dbid	*dbid* of testVia.
unusedPins	yes	t/nil	Allows test points on pins, not on a net.
	The following are Text Controls		
alpha	yes	t/nil	If `t`, use Alphabetic extension, `nil` use numeric extension.
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.

3

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.

axlGetParam

```
axlGetParam (
     t_parm_name
)
⇒ •o_paramDbid/nil
```

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

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

`paramTextBlock:<#>` where # is 1-16 (Example: `paramTextBlock:1`)
`paramDesign`
`paramDisplay`
`paramLayerGroup:<name>` where name is legal Allegro class name
`paramLayerGroup:<name>/paramLayer:<name>`

`artwork`List of film names

`artwork:<filmName`A film given by `filmName`

`testprep`See `axlParamTestPrepDoc`

Value Returned

o_paramDbid	*dbid* for the requested parameter.
`nil`	Parameter requested not found.

Example

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

```
Skill> etch_parm = axlGetParam("paramLayerGroup:ETCH")
param:123456
```

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

Skill> etch_parm->color
-1

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

2) Access artwork records:

A) Get list of all possible records.

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

B) Get information on film record “VCC”.

```
r = axlGetParam("artwork:VCC")
Skill> 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"
)
```

C) Delete a TOP parameter record.

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

3) Design (`paramDesign`) modification.

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

See Also

Description of Database Objects on page 70.

axlSetParam

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

Description

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

Arguments

o_paramDbid	Parameter id returned from axlGetParam. Modifying the contents of a record retrieved from GetParam changes an existing parameter.

Value Returned

o_paramDbid	Returns the input parameter id if successful
nil	Database was not modified.

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`
- `'multivalue` - use `dfor` fields where value not the same
- `'button` - current color of button faces (grey)

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

AXL API calls such as `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.

Example:

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

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

Arguments

none

Value Returned

none

axlColorGet

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

Description

Get color palette. Supports the following modes:

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

Arguments

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

Value Returned

x_count	Size of database palette.
`nil`	Error.
lx_rgb	A palette.

llx_rgb The entire database palette.

Examples

Get red/green/blue of color 2:

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

Get background color:

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

Get number of colors:

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

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

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

axlColorShadowGet

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

Description

Provides the options of shadow mode.

Arguments

g_option

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

Value Returned

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

See Also

axlColorSet, axlColorShadowSet

Examples

Is shadow mode on:

```
axlColorShadowGet('mode)
```

Is shadow mode percent:

```
axlColorShadowGet('percent)
```

axlColorShadowSet

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


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

Description

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

Options are:

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

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

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

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

Arguments

'mode	Enable or disable shadow mode.
'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.
```

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

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

Value Returned

t	If loaded file.
nil	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.

axlColorOnGet

```
        x_colorNumber
) -> g_state/nil


        'count
) -> x_count


        'all
) -> lg_state

        lx_colorNumber
) -> lg_state
```

Description

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

Modes are:

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

Arguments

x_colorNumber	Color number. Value between one and maximum colors.
lx_colorNumber	List of color numbers.
'count	Return maximum colors.
'all	Returns visibility state of all colors.

Value Returned

g_state	`t` color is on, `nil` color is off.

x_count	Total number of colors in design.
lg_state	List of visibility state for all colors.
nil	nil return may indicate an error.

Examples

If color number 2 is on:.

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

Get number of colors:

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

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

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

Get color numbers

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

See Also

axlColorSet, axlColorOnSet

axlColorOnSet

```
        x_colorNumber/lx_colorNumber
        g_state
) -> t/nil


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

Description

Sets the visibility of colors. Modes are:

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

Arguments

`x_colorNumber`	Color number between one and maximum colors.
`lx_colorNumber`	List of color numbers.
`g_state`	`t` for on and `nil` for off.
`'all`	All color numbers.

Value Returned

`t`	Success.
`nil`	Illegal value provided.

Examples

Set color number three to on:

```
axlColorOnSet(3 t)
```

Set color number all colors off:

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

Set color number six, seven, eight to on:

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

See Also

axlColorSet, axlColorOnGet, `axlVisibleUpdate`

axlColorPriorityGet

```
        'all

) -> lx_colorNumbers


        x_colorNumber

) -> x_priority/nil


        'priority

        x_priority

) -> x_colorNumber/nil
```

Description

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

The modes supported by this function are:

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

Arguments

'all	Return color priority table.
'priority	Argument is the color number.
x_colorNumber	Obtain priority of given color number.

x_priority	Obtain color at given priority.

Value Returned

x_priority	Priority of given color.
x_colorNumber	Color number at given priority.
lx_colorNumbers	Color priority table.
nil	Error based on color number or priority number greater then maximum colors.

Examples

Return priority of all colors:

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

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

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

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

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

See Also

axlColorPrioritySet, axlColorSet

axlColorPrioritySet

```
    lx_colornumbers
) -> t/nil

    'top
    x_colorNumber
) -> t/nil

    'change
    x_oldPriority
    x_newPriority
) -> t/nil

    x_colorNumber
    x_newPriority
) -> t/nil
```

Description

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

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

Arguments

'all	Returns color priority table.

'priority	Next argument is the color number.
x_colorNumber	Obtain priority of given color number.
x_priority	Obtain color at given priority.

Value Returned

t	Success.
nil	Error in one of the arguments.

Examples

Set priority of all colors:

```
axlColorPrioritySet('all)
```

Set color number two at top priority:

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

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

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

Assign color four priority one:

```
axlColorPrioritySet(4 1 )
```

See Also

axlColorPriorityGet, axlColorPrioritySet, axlVisibleUpdate

axlColorSave

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

Description

Saves current design colors to specified file.

Argument

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

Value Returned

`t`	Successful.
`nil`	Failed to save.

EXAMPLES

Save current design color settings:

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

See Also

axlColorSave,`axlColorSet`

axlColorSet

```
axlColorSet(

                  x_number/'background

                  l_rbg

            ) -> t/nil

axlColorSet(

                  'all

                  ll_rgb

            ) ->t/nil
```

Description

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

Modes supported:

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

Red/green/blue colors are values between `0` (least intesity) to `255` (maximum 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.
- The option of color on or off (this state is not saved with the database) provided by the checkbox in the color priority command.

- Drawing priority where one is the highest priority. No two colors may have the same priority. Use the color boxes in the color priority command to rearrange the colors.
- Each color number can be assigned to a layer. Multiple layers will have the same color number, because there are more layers than colors, .
- 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.
`axlColorGet`	Get red, green, or blue of one or more color numbers.
`axlColorOnGet`	Get if color number is turned on or off.
`axlColorOnSet`	Set color number on or off.
`axlColorPrioritySet`	Control drawing priority of color numbers.
`axlColorPriorityGet`	Provides the color priority.
`axlColorShadowGet`	Shadow mode options.
`axlColorShadowSet`	Set shadow mode options.
`axlColorSave`	Save color values to file.
`axlColorLoad`	Load color values from file.
`axlUIColorDialog`	Standard color chooser dialog box.
`axlDBControl`	Miscellaneous color number assignments (for example, highlight).
`axlLayerGet`	Get layer (class/subclass) attributes (control color) number and visibility for individual layers.
`axlLayerSet`	Set color number or visibility for a layer.
`axlVisibleLayer`	Set visibility of layer.
`axlIsVisibleLayer`	Provides the layer visibility.

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

Arguments

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

Value Returned

`t`	Successful.
`nil`	An error; wrong arguments: color number is less then 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]
         )
         ==> t/nil
```

Description

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

Arguments

x_custom_color	color index to initialize palette dialog. values 0 to 293.
g_show_hilite	if highlight checkbox is to be displayed
	t display highlight checkbox.
	nil/default - do not.
x_hilite_flag	Highlight state to initialize highlight checkbox (if displayed). Pass 1 or 0.

Value Returned

list	containing one or two int values for user color palette selection and highlight checkbox selection. if `g_show_hilite` is not `nil`, list contains the two values, or else list contains color index only.
nil	if user cancels the form or error occured.

axlClearObjectCustomColor

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

Description

Clear custom color of dbids

Arguments

lo_dbid:	List of dbids to clear custom color.

Value Returned

t/nil:	Returns `t` if at least one object custom color was cleared. Returns `nil` otherwise.

Examples

See axlCustomColorObject for examples

See Also

axlCustomColorObject

axlCustomColorObject

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

Description

Custom color the provided dbid or list of dbids.

Arguments

od_dbid	list of DBIDS or one DBID
g_custom_color	color index to be used to set custom color. if nil, perm highlight will be used.

Value Returned

t	Something was custom colored.
nil	No valid dbids.

See Also

axlClearObjectCustomColor, axlDBControl, axlIsCustomColored

Example

You can use the AXL-SKILL `axlCustomColorObject` and `axlClearObjectCustomColor` functions to set/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.

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`If die stack layer.

`nil` If not.

See Also

`axlDBIsBondwire`

Examples

```
axlDBIsDieStackLayer("CONDUCTOR/TOP_COND") -> nil
```

axlGetDieData

```
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 SIP products.

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 fields:

`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

Example

```
data = axlGetDieData("FLIPCHIP_1")
printf("stack-pos = %L, layer-name = %L, attach-type  = %L\n"
          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 SIP products.

Arguments

`g_stackArg` name or *`dbid`* 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_TOP, DSA_SUBSTRATE_BOTTOM, DSA_SUBSTRATE_CAVITY_TOP, DSA_SUBSTRATE_CAVITY_BOTTOM

`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 SIP products.

Arguments

`g_stackArg` name or *`dbid`* of the die stack

Value Returned

`l_data`List of die-stack member data defstructs

`nil`If error

defstruct fields:

`dbid` *`dbid`* 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
```

```
          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 ICP products.

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 SIP products.

Arguments

`g_iposerName` name or *`dbid`* 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`*`dbid`* 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

`dielMatl`name of dielectric material used for substrate

`dielThickness`thickness of dielectric material

`condMatl`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 SIP products.

Arguments

`g_spacerName` name or *dbid* 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`*`dbid`* 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 occured 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.

axlLayerPriorityClearAll

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

Description

Clears all layer priority information in Allegro database. Use axlLayerPrioritySet() for usage.

Arguments

None

Value Returned

t:	success

See Also

axlLayerPrioritySaveAll

axlLayerPriorityRestoreAll

axlLayerPriorityRestoreAll

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

Description

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

Aarguments

None

Value Returned

t:	success
nil:	nothing to restore.

See Also

axlLayerPrioritySaveAll

axlLayerPriorityClearAll,

axlLayerPrioritySaveAll

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

Description

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

Arguments

None

Value Returned

t:	success
nil:	this function has been called already but axlLayerPriorityRestoreAll has not been called yet.

See Also

axlLayerPriorityClearAll, axlLayerPriorityRestoreAll

axlLayerPrioritySet

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

Description

This changes the priority of given layer.

Arguments

x_layer:	layer name (i.e. "ETCH/TOP")
x_priority:	priority value in the range of 1-255.

Note: priority value of 0 means remove layer priority.

Value Returned

t:	success
nil:	error in one of the arguments

Examples

Set prirority for class GEOMETRY and subclass OUTINE:

```
axlLayerPrioritySet("BOARD_GEOMETRY/BGEO_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()

It's your responsibility to call db_SaveColorTableLayerPriority() to restore old color table.

See Also

axlLayerPriorityClearAll, axlLayerPrioritySaveAll, axlLayerPriorityRestoreAll

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 color or nil if object has no custom color or object does not support custom color.

See Also

axlCustomColorObject

axlSetDieData

```
axlSetDieData (
          g_dieId
          s_dataType
          g_newValue
)
==> t/nil
```

Description

Sets the given data for the given die.

Only available in SIP products.

Arguments

`g_dieName`name or *`dbid`* 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`

axlSetIposerData

```
axlSetIposerData (
          g_iposerId
          s_dataType
          g_newValue
)
==> t/nil
```

Description

Sets the given data for the given iposer.

Only available in SIP products.

Arguments

`g_iposerName`name or *`dbid`* 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 SIP products.

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).

Database Layer Management

These functions allow easier access to layer attributes.

axlDBGetLayerType

```
axlDBGetLayerType
     t_layerName

)

⇒ t_layertype/nil
```

Description

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

Arguments

t_layername	Layername is <*class*>/<*subclass*>.

Value Returned

t_layertype	Layer type string.
`nil`	Layer is invalid.

Example

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

axlGetXSection

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

Description

Returns a list of all layers in the current drawing's cross section.

Notes

Both *t_SignalDieConstant* and *t_SignalLossTangent* are the values for dielectric material between traces on signal layers. Beginning with 15.0, a property stores the *dieConst/LossTan* for dielectric material on signal layers.

The dielectric material properties from the dielectric layer above (toward the closest surface) is used as the dielectric material between signal traces unless the design property SQ_STRIPLINE_DIELECTRIC is set. Then this defines the material, dielectric constant, and loss tangent to be used for non-plane, non-surface conductor layers. Surface layers always look to the outside for their dielectric.

If the property is missing, invalid, or the material in the property is set to _LS_AUTOMATIC_, then the *t_SignalDieConstant* and *t_SignalLossTangent* values are set to match those of the adjacent dielectric (looking toward the closest surface).

New releases may add to the list of data returned for each layer. These items will be added at the end of the list. Added items:

```
16.01 - g_freqDepFileName
```

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:

t_name Layer name.

t_type	Layer type.
t_material	Layer material.
t_thickness	Layer thickness.
t_thermalCond	Layer thermal conductivity.
t_elecCond	Layer electrical conductivity.
t_dielectricConst	Layer dielectric constant.
y_artworkNeg	Indicates whether the artwork for the layer is negative.
y_shield	Indicates whether the layer is a shield layer.
t_lossTangent	Layer loss tangent (valid for dielectrics only).
t_usage	Layer usage (that is, IC RDL or DRIVER layers)
t_SignalDieConstant	Dielectric between traces on interior signal layers (or *nil*).
t_SignalLossTangent	Dielectric between traces on interior signal layers (or*nil*).
g_freqDepFileName	Defines the name of the frequency-dependent data file for the file; *nil* if no file name is defined for this layer.

Note: The *t_SignalDieConstant* and *t_SignalLossTangent* are *nil* on PLANE and dielectric layers. If an error occurs, returns *nil*.

axlIsLayer

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

Description

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

Arguments

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

Value Returned

t	Layer exists.
nil	Layer does not exist.

axlIsVisibleLayer

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

Description

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

Arguments

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

Value Returned

`t`	Layer is visible.
`nil`	Layer is invisible or not present.

Example

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

axlLayerCreateCrossSection

```
axlLayerCreateCrossSection(
     t_Prev_layerName
     t_layerType
     t_materialType
     [t_subclassName]
     [t_planeType]

)

⇒ t/nil
```

Description

Adds a new cross-section layer to the design.

Arguments

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

Value Returned

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

Example

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

Creates a conductor layer with these characteristics:

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

axlLayerCreateNonConductor

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

Description

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

Arguments

t_layerName	<*class*>/<*subclass*>

Value Returned

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

Example

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

Creates a new subclass named MYSUBCLASS.

axlLayerGet

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

Description

Gets the layer parameter given the shortcut notation of <*class*>/<*subclass*>.

See Also

axlSetParam

Arguments

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

Value Returned

o_dbid	Layer parameter *dbid*.
nil	Layer is not present.

Example

Changes color of top etch layer.

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

axlVisibleDesign

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

Description

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

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

```
saveVis = axlVisibleGet()
axlVisibleDesign(nil)
; set desired layers visible via one or more calls to
axlVisibleLayer(...)
; set find filter for objects to find
axlSetFindFilter(...)
; find objects by using one of the Select APIs .. example
axlAddSelectAll()
objs = axlGetSelSet()

; restore visiblility
axlVisibileSet(saveVis)
; note no need to make a call to axlVisibileUpdate 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)
("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 visiblity updates.

Arguments

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

Value Returned

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

axlDBCreateFilmRec

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

Description

Creates a film record with parameters and visible layers specified.

Example 1 on page 182 shows a common use of this function through the film control form's film record *load* option, which calls `axlfcreate` using the following form:

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

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

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

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

Arguments

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

Value Returned

t_name	Name of the film record created.
nil	Failure to create the film record.

Example 1

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

- Called through film control form

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

`axlfcreate` changes layer visibility and calls `axlDBCreateFilmRecord` as shown:

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

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

Example 2

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

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

axlSetPlaneType

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

Description

Changes the plane type of the conductor subclass.

Arguments

t_subclassName	Subclass name whose plane type is to be changed.
t_planeType	Plane type (Positive, Negative)

Value Returned

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

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/SIP as certain class names are different in these products.

The base call is actually just:

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

Arguments

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

Values Returned

lt_subclasses	A list of subclass strings.

See Also

axlSubclassRoute, axlGetParam, and axlMapClassName

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")
```

4

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. **
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 223 for details on how to use these keywords:

`Global`	`ALL`, `ALLTYPES`, `EQUIVLOGIC`
`Figures`	`PINS`, `VIAS`, `CLINES`, `CLINESEGS`, `LINES`, `LINESEGS`, `DRCS`, `TEXT`, `SHAPES` (includes rectangles), `SHAPESEGS`, `VOIDS`, `VOIDSEGS`, `TEXT`. **Note:** The *`xxx`*`SEG` keywords also select arc and circle segments
`Logic`	`COMPONENTS`, `SYMBOLS`, `NETS`

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

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

Selection and Find Functions

This section lists selection and find functions.

axlSingleSelectPoint

```
axlSingleSelectPoint(
     [l_point]

)

⇒ t/nil
```

Description

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

`axlSingleSelectPoint` selects a *single* object and adds it to the select set, unless `EQUIVLOGIC` is on. In that case, it may select multiple objects, for example, if it finds a qualified figure (such as a pin, connect line, or via) that is part of a net. `axlSingleSelectPoint` adds all the qualified figures that belong to the net to the select set. (See the Find Filter Options on page 194 .)

This finds objects within the current trapsize (`axlGetTrapBox`), which varies based upon the zoom factor.

Arguments

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

Value Returned

`t`	One or more *dbids* put in the select set.
`nil`	No *dbids* put in the select set.

Example

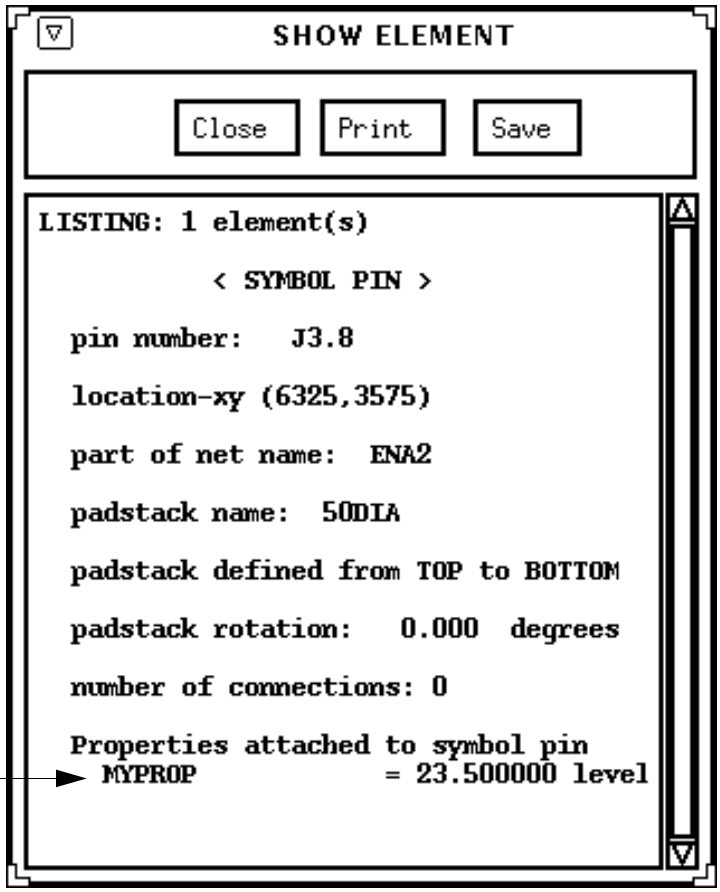
```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")
     axlSingleSelectPoint( 6325:3575 )
     axlDBAddProp(axlGetSelSet() list( "MYPROP" 23.5))
     ⇒t
```

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

To check

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

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



axlAddSelectPoint

```
axlAddSelectPoint(
     [l_point]

)

⇒ t/nil
```

Description

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

Selects a *single* object and adds it to the select set, unless EQUIVLOGIC is on. In that case, selects multiple objects, for example, if it finds a qualified figure (such as a pin, connect line, or via) that is part of a net. Adds all the qualified figures that belong to the net to the select set. (See the Find Filter Options on page 194 .)

This finds objects within the current trapsize (axlGetTrapBox), which varies based upon the zoom factor.

Arguments

l_point	Point in the layout at which to find figures.

Value Returned

t	One or more *dbids* put in the select set.
nil	No *dbids* put in the select set.

Example

See axlSingleSelectPoint on page 195 for an example. axlSingleSelectPoint has the same behavior except that it selects only one object.

axlSubSelectPoint

```
axlSubSelectPoint(
     [l_point]

)

⇒ t/nil
```

Description

Finds a figure at *l_point* according to the Find Filter and deletes its *dbid* from the select set in cumulated mode. That is, it deletes that *dbid* while leaving any others currently in the select set. If *l_point* is nil, requests a single pick from the user.

Removes a *single* object from the select set, unless EQUIVLOGIC is on. In that case, finds multiple objects, for example, if it finds a qualified figure (such as a pin, connect line, or via) that is part of a net. Deletes all the qualified figures that belong to the net from the select set. (See the Find Filter Options on page 194 .)

This finds objects within the current trapsize (axlGetTrapBox), which varies based upon the zoom factor.

Arguments

l_point	Point in the layout at which to find figures to deselect.

Value Returned

t	One or more *dbids* removed from the select set
nil	No *dbids* removed from the select set.

Example

```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")
axlSingleSelectBox( list(6200:3700 6500:3300))
axlSubSelectPoint( 6325:3575 )
axlDBAddProp(axlGetSelSet() list("MYPROP" 23.5))
⇒ t
```

Adds a previously defined user property, MYPROP

1. Selects four pins in a box in order.

2. Before adding the property, subtracts the pin at `6325:3575` from the list, then adds as shown.

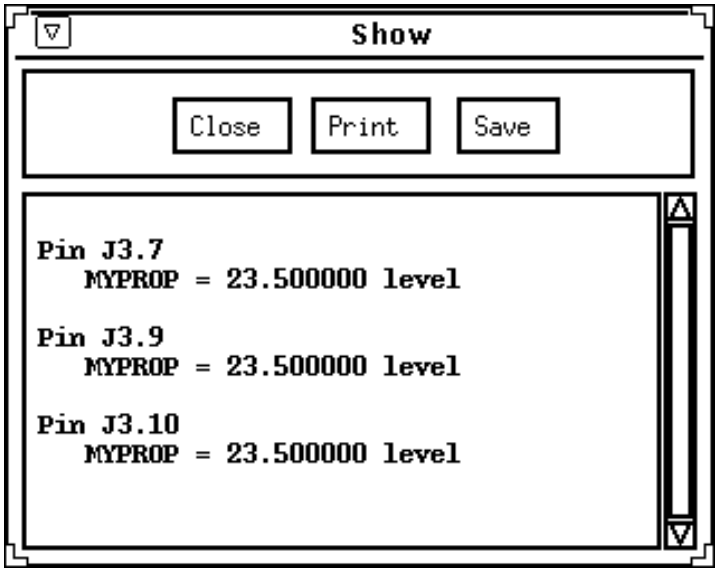
To check

1. Select *Edit – Properties*.

2. Select a window around all four pins.

3. Select *MYPROP* from the Available Properties list in the **Edit Property** window.

The command displays the pins that have properties in the **Show Properties** window. Only the three pins not subtracted from the select set have the property, *MYPROP*.



axlSingleSelectBox

```
axlSingleSelectBox(
     [l_bBox]

)

⇒ t/nil
```

Description

Clears the select set, finds all figures inside the rectangle *l_bBox* according to the Find Filter, and adds the selected figure *dbids* in single mode to the select set.

Arguments

l_bBox	List containing one or two coordinates defining the bounding box to be used to select the figures. If *l_bBox* is `nil`, requests two picks from the user. If *l_bBox* has only one point, asks for a second point from the user.

Value Returned

`t`	One or more objects added to the select set.
`nil`	No objects added to the select set.

Example

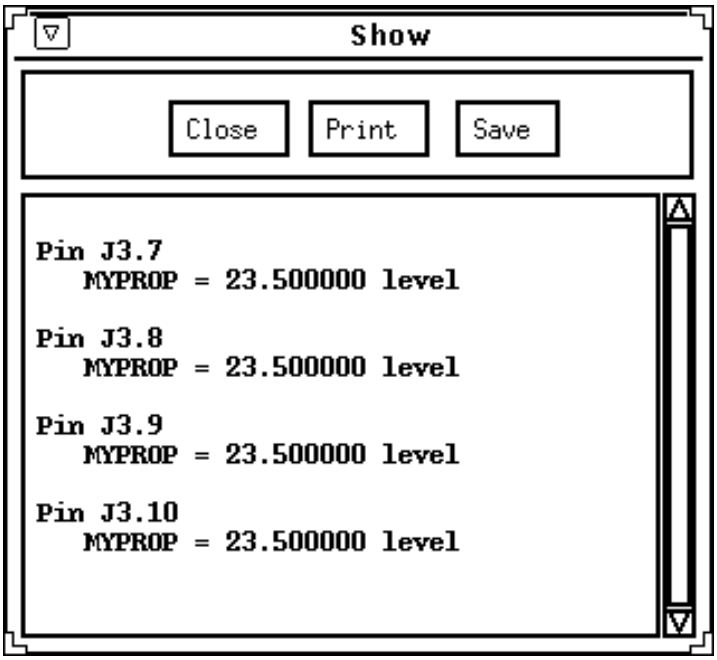
```
axlSetFindFilter(?enabled "pins" ?onButtons "pins")
axlSingleSelectBox( list(6200:3700 6500:3300))
axlDBAddProp(axlGetSelSet() list( "MYPROP" 23.5))
⇒ t
```

Adds a previously defined user property `MYPROP` to four pins by selecting a box around them with corners (`6200:3700 6500:3300`).

To check

1. Select *Edit – Properties*.
2. Select the four pins.
3. Select *MYPROP* from the Available Properties list in the **Edit Property** window.

The command displays the four pins in the **Show Properties** window as having *MYPROP* with value *23.500000 level*



axlAddSelectBox

```
axlAddSelectBox(
     [l_bBox]
)
⇒ t/nil
```

Description

Finds one or more figures inside the rectangle *l_bBox* according to the current Find Filter, and adds the selected figure *dbids* in cumulated mode for the select set.

Arguments

l_bBox	List containing one or two coordinates defining the bounding box to be used to the select figures. If *l_bBox* is `nil`, requests two picks from the user. If *l_bBox* has only one point, asks for a second point from the user.

Value Returned

`t`	One or more objects added to the select set.
`nil`	No objects added to the select set.

Example

See the example for axlSingleSelectBox on page 200. That function behaves exactly as `axlAddSelectBox`, except that `axlSingleSelectBox` does not clear the select set.

axlSubSelectBox

```
axlSubSelectBox(
     [l_bBox]
)
⇒ t/nil
```

Description

Finds one or more figures inside the rectangle *l_bBox* according to the Find Filter, and deletes their *dbids* from the select set in *cumulated* mode. Deletes those *dbids* while leaving any others currently in the select set.

Arguments

l_bBox	List containing one or two coordinates defining the bounding box to be used to the select figures. If *l_bBox* is `nil`, requests two picks from the user. If *l_bBox* has only one point, asks for a second point from the user.

Value Returned

`t`	One or more objects deleted from select set.
`nil`	No objects deleted from select set.

Example

See axlSubSelectPoint on page 198 for an example of subtracting from the select set, and axlAddSelectPoint on page 197 for an example of using a box for selection.

axlAddSelectAll

```
axlAddSelectAll(

)

⇒ t/nil
```

Description

Finds all the figures in the database that pass the current Find Filter and adds their *dbids* to the select set.

Arguments

None.

Value Returned

`t`	One or more object *dbids* added to the select set.
`nil`	No object *dbids* 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 *dbids* deleted from select set.
`nil`	No *dbids* 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

t_nameType	Indicates the type of name string being provided. Also implies the type of object to be selected. (see Finding Objects by Name on page 190).

l_names	One of three possibilities (See examples):
	`-Name`
	`-List of names`
	`-List of name/value pairs`

g_wildcard	A `t` means that `*` or `?` performs regular expression matching. Default is `nil` 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

t_nameType	String denoting the name type to be selected (see Finding Objects by Name on page 190). Also implies the type of object to be selected.
l_names	One of three possibilities (See examples): A name A list of names A list of name/value pairs

Value Returned

`t`	One or more objects added to the select set.
`nil`	No objects added to the select set.

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 207. 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

t_nameType	String denoting name type to be selected (see Finding Objects by Name on page 190). Also implies the type of object to be selected.
l_names	One of three possibilities (See examples): -Name -List of names -List of name/value pairs

Value Returned

`t`	One or more objects deleted from select set.
`nil`	No objects deleted from select set.

Note: 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 207. The only difference is that `axlSingleSelectName` clears the select set and then puts the *dbids* it finds into the

select set, while `axlSubSelectName` removes the *`dbids`* of the elements it selects from the select set. The arguments are the same as `axlSingleSelectName`.

axlSingleSelectObject

```
axlSingleSelectObject(
     lo_dbid

)

⇒ t/nil
```

Description

Clears contents of the select set and adds the *dbids* in *lo_dbid* to the select set in single mode. *lo_dbid* is either a single *dbid* or a list of *dbids*. Selects figures completely regardless of the selection mode. If the *dbid* of any part of a figure is in *lo_dbid*, the function adds the entire figure.

Arguments

lo_dbid	*dbid*, or list of *dbids* to be added to the select set.

Value Returned

t	One or more objects added to the select set.
nil	No objects added to the select set.

Example

```
axlClearSelSet()
axlSetFindFilter(
     ?enabled list( "all" "equivlogic")
     ?onButtons list( "all"))
mysym = axlDBCreateSymbol(
     list( "dip14" "package"), 5600:4600)
axlSingleSelectObject(car(mysym))
⇒ t
```

Creates a symbol and add its *dbid* to the select set.

axlAddSelectObject

```
axlAddSelectObject(
     lo_dbid

)

⇒ t/nil
```

Description

Adds the *dbids* in *lo_dbid* to the select set in cumulated mode, that is, without removing already selected objects. *lo_dbid* is either a single *dbid* or a list of *dbids*. Adds *dbids* in the select set only if they are not already there. Selects figures completely regardless of the selection mode. If the *dbid* of any part of a figure is in *lo_dbid*, adds the entire figure.

Arguments

lo_dbid	*dbid*, or list of *dbids* to be added to the select set.

Value Returned

t	One or more objects added to the select set.
nil	No objects added to the select set.

Example

```
axlSetFindFilter(
     ?enabled list( "all")
     ?onButtons list( "all"))
mysym = axlDBCreateSymbol(
     list("dip14" "package") 2600:2600 ) axlAddSelectObject(car(mysym))
⇒ t
```

Creates a symbol instance and adds its *dbid* to the select set.

axlSubSelectObject

```
axlSubSelectObject(
     lo_dbid
)
⇒ t/nil
```

Description

Removes the *dbids* in *lo_dbid* from the select set in cumulated mode. *lo_dbid* is either a single *dbid* or a list of *dbids*. Removes figures completely regardless of the selection mode.

Arguments

lo_dbid	*dbid*, or list of *dbids* to be removed from select set.

Value Returned

t	One or more objects deleted from select set.
nil	No objects deleted from select set.

Example

```
axlClearSelSet()
axlSetFindFilter(?enabled list("noall" "vias")
     ?onButtons list( "vias"))
myvia = axlDBCreateVia( "pad1", 3025:3450)
axlAddSelectBox( list( 3000:3100 3200:3600))
axlSubSelectObject( car( myvia))
⇒ t
```

Creates a via, then selects all the vias in a surrounding region for deletion, but saves the one just created by subtracting its *dbid* from the selection set.

The resulting select set contains the *dbids* of all vias in the box except *myvia*, the one just created.

axlSelect

```
axlSelect(
     ?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

`firstEventCallback`	Procedure to be called once the first user event occurs. The callback takes no arguments.

groupMode	Default is for `axlSelect` to return when the user makes a selection. If groupMode is `'t`, `axlSelect` won't return until you do an `axlCancelEnterFun` or `axlFinishEnterFun`. You perform all of your activity in popup callbacks instead of when `axlSelect` returns. In `groupMode`, `axlSelect` does not clear existing selections. To clear existing selections after the first event, clear them in your *`firstEventCallback`*.
prompt	Prompt to the user. The default prompt is: `"Enter selection point"`

Value Returned

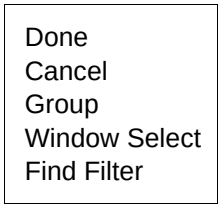
`t`	One or more object *`dbids`* put into the select set during the call. The select set is a list of the *`dbids`* of the objects the user selected.
`nil`	No object *`dbids`* put into the select set.

Example

```
(defun showElement ()
mypopup = axlUIPopupDefine( nil
     (list (list "Done" 'axlFinishEnterFun)
     (list "Cancel" 'axlCancelEnterFun)))
axlUIPopupSet( mypopup)
axlSetFindFilter( ?enabled list( "NOALL" "ALLTYPES" "NAMEFORM")
     ?onButtons "ALLTYPES")
while( axlSelect()
     axlShowObject( axlGetSelSet()))
```

Function loops, performing the `Show Element` command on each object selected by the user.

Although you explicitly set *Done* and *Cancel* for this command, AXL also adds *Group*, *Window Select*, and *FindFilter* to the pop-up.



axlGetSelSet

```
axlGetSelSet(
)
⇒ lo_dbid/nil
```

Description

Gets the list of object *dbids* in the select set.

Arguments

None.

Value Returned

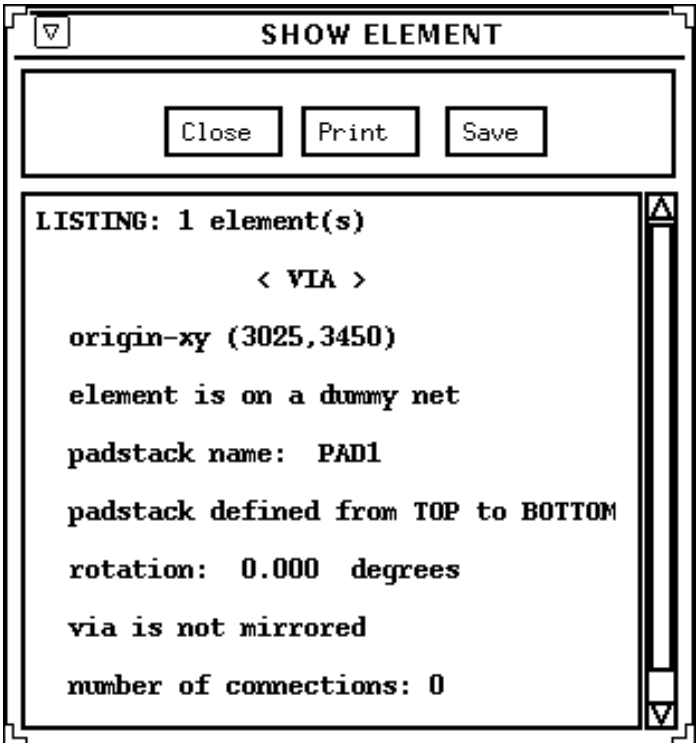
lo_dbid	List of figure *dbids*.
nil	Select set is empty.

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.

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 *dbids* removed in order to empty the select set.
`nil`	Select set already empty.

Example

```
axlClearSelSet()
axlSetFindFilter(?enabled list("noall"))
axlSetFindFilter(?enabled list("pins")
     ?onButtons list("pins"))
axlAddSelectBox()
axlGetSelSetCount()
⇒ 14
```

Ensures the select set does not have any leftover *dbids* in it, as in the example for `axlGetSelSetCount`.

axlGetFindFilter

```
axlGetFindFilter(
     [onButtonF]
)
⇒ lt_filters/nil
```

Description

Gets the current Find Filter settings and returns them as keyword strings in the list lt_filters *lt_filters* according to the value of *onButtonF*, as described below.

Arguments

onButtonF	If *onButtonF* is t, returns the enabled list. If *onButtonF* 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"))
(axlGetFindFilter)
```

Returns the following:

```
("NAMEFORM" "NETS" "CLINESEGS" "VIAS" "PINS")
```

axlSetFindFilter

```
axlSetFindFilter(
     ?enabled lt_enabled
     ?onButtons lt_filterOn

)

t/nil
```

Description

Sets up both the object types to be displayed in the Find Filter, and which types among those are set to *on* in the **Find Filter** .

The first argument, *lt_enabled*, is a list of the object types to be displayed in the Find Filter and of the select options described. The second argument, *lt_onButons*, 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 enble all object types except for pins.

```
'("ALLTYPES" "NOPINS")
```

Each of the following keywords may be preceeded with a "NO" to disable the particular option or object type. For example, "NOPINS". The initial default is "NOALL".

`axlSetFindFilter` Keywords

Keyword	Description
"PINS"	Enable pins
"VIAS"	Enable vias
"CLINES"	Enable clines
"CLINESEGS"	Enable cline (arc or line) segs
"LINES"	Enable lines
"LINESEGS"	Enable line (arc or line) segs
"DRCS"	Enable drc errors
"TEXT"	Enable text
"SHAPES"	Enable shapes, rects and frects

`axlSetFindFilter` Keywords, *continued*

Keyword	Description
`"SHAPESEGS"`	Enable shape segments
`"BOUNDARY_SHAPES"`	Enable promotion to boundary shape if auto-shape is selected (see dynamic shape discussion)
`"VOIDS"`	Enable shape voids
`"VOIDSEGS"`	Enable shape void segments
`"SYMBOLS"`	Enable symbol instances
`"FIGURES"`	Enable figures
`"COMPONENTS"`	Enable component instances
`"FUNCTIONS"`	Enable function instances
`"NETS"`	Enable nets
`"AUTOFORM"`	Option - OBSOLETE
`"EQUIVLOGIC"`	Option - For logic object types (nets and components), causes the physical equivalent to be selected. Physical objects must be enabled in both "*`enabled`*" and "*`onButton`*" lists.
`"INVISIBLE"`	Option - Allows selection of objects that are not visible due to color *`class/subclass`* form setting.
`"NAMEFORM"`	Option - Enables the *Find by Name/property* fields in the Find Filter form.
`"ALLTYPES"`	Enables all object types `("PINS" ... "NETS")`.
`"ALL"`	Enables all object types and options.
`"DYNTHEMALS"`	Enable selection of themal reliefs generated by dynamic shapes. Only applicable to `?enabled`. By default, you should not select these if you plan on modifying the objects since the dynamic shape will just re-generate them. You should only access them for read-only purposes.

`axlSetFindFilter` Keywords, *continued*

Keyword	Description
`"GROUPS"` `"GROUPMEMBERS"`	`"GROUPS"` and `"GROUPMEMBERS"` operate together to produce four possible selection states: **Keyword** / **States** 1 2 3 4 GROUPS: OFF ON OFF ON GROUPMEMBERS: OFF OFF ON ON ■ State 1: Legacy. This supports code that predates the group implementation. This is the same as State 3. ■ State 2: Group only. By only setting the group bitfield, any selected group is returned to the application as a group. ■ State 3: Members only. By only setting the group_members bitfield, group members are returned to the application when a group is selected. ■ State 4: Hierarchical. By setting both the group and group_members bitfields, a group is returned for any heirarchical group that is selected (such as a Module instance), and group members are returned for all other selected group types.
`"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" an "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*

Keyword	Description
`"DYNTHEMALS"`	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 modifing the objects since the dynamic shape will just re-generate them. You should only access them from read-only purposes. If in the partition editor or the design has partitions active then this option is used to selections of read-only objects. This option should also be used to select shape base fillets. The sytem will not allow shape based fillets to be modified it the dynamic fillet option is enabled.
`"BONDSMART"`	Option - Application has been updated to differentiate bond wires and/or fingers (APD/SIP). For more information, see Bond Objects.
`"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. If your SKILL program does not access shapes, read no further.

A shape on `ETCH` may be either static or dynamic. A static shape is similar to what existed in Allegro PCB Editor prior to PSD15.0. You add a shape to an etch layer and manually void objects that impact the shape. A dynamic shape is placed on the `BOUNDARY CLASS` and generates zero or more shapes on the `ETCH` layer based upon voiding.

If you want to modify a dynamic shape, then you should set `BOUNDARY_SHAPES`. This allows the user to select the generated shape, but selection will return its dynamic shape (e.g. a

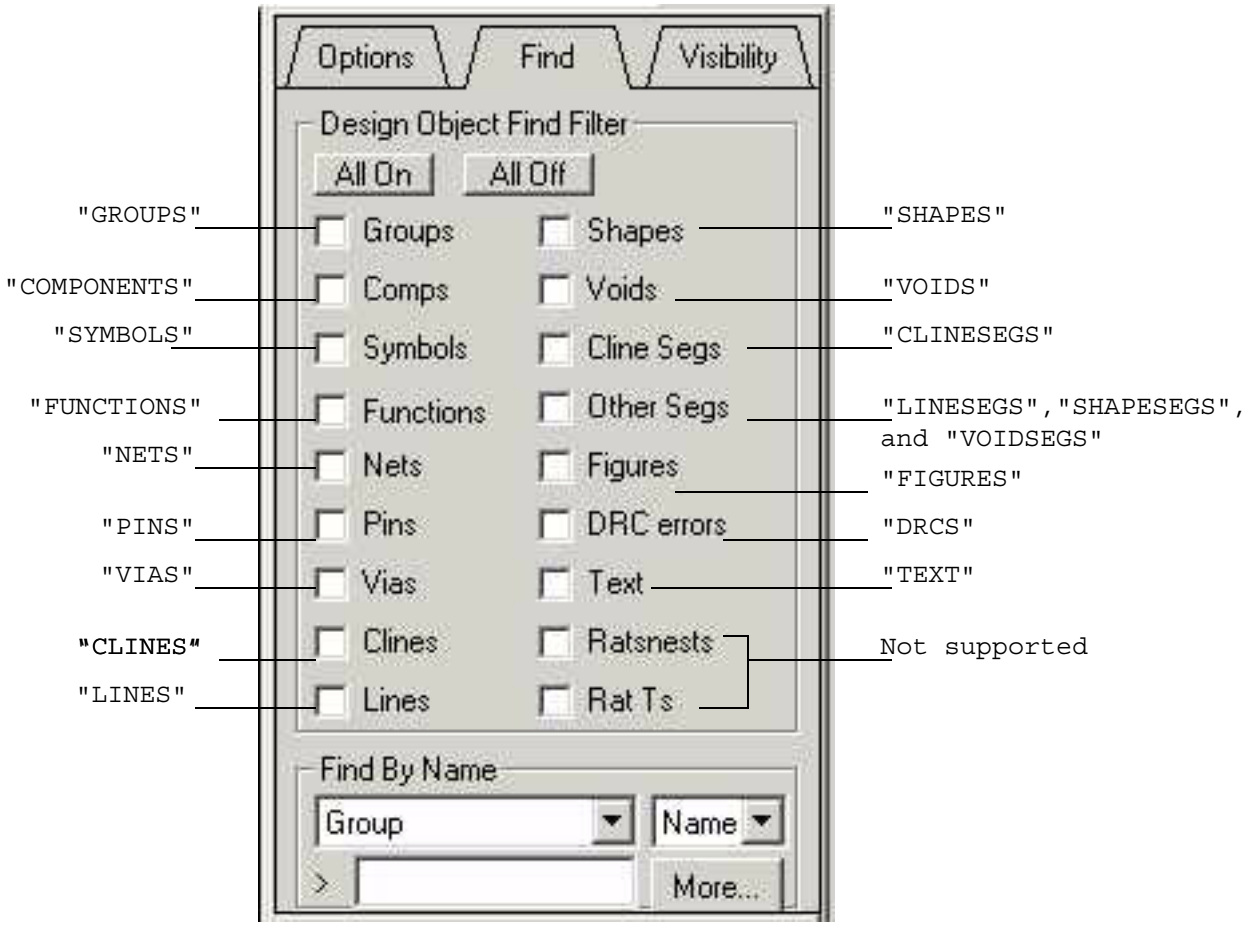
move shape). If you want to access information on the shape, then do not set the `BOUNDARY_SHAPES` option (e.g. *show element*).

Note: If you pass `"ALL"` or `"ALLTYPES"` to *`setOptions`*, then `BOUNDARY_SHAPES` will be enabled and user's selecting a `ETCH` layer generated shape will result in the selelction returning its dynamic shape on the `"BOUNDARY CLASS"`. If you wish to select the generated shape, but want to use the `all` option, then use the following:

```
"(ALLTYPES" "NOBOUNDARY_SHAPES")
```

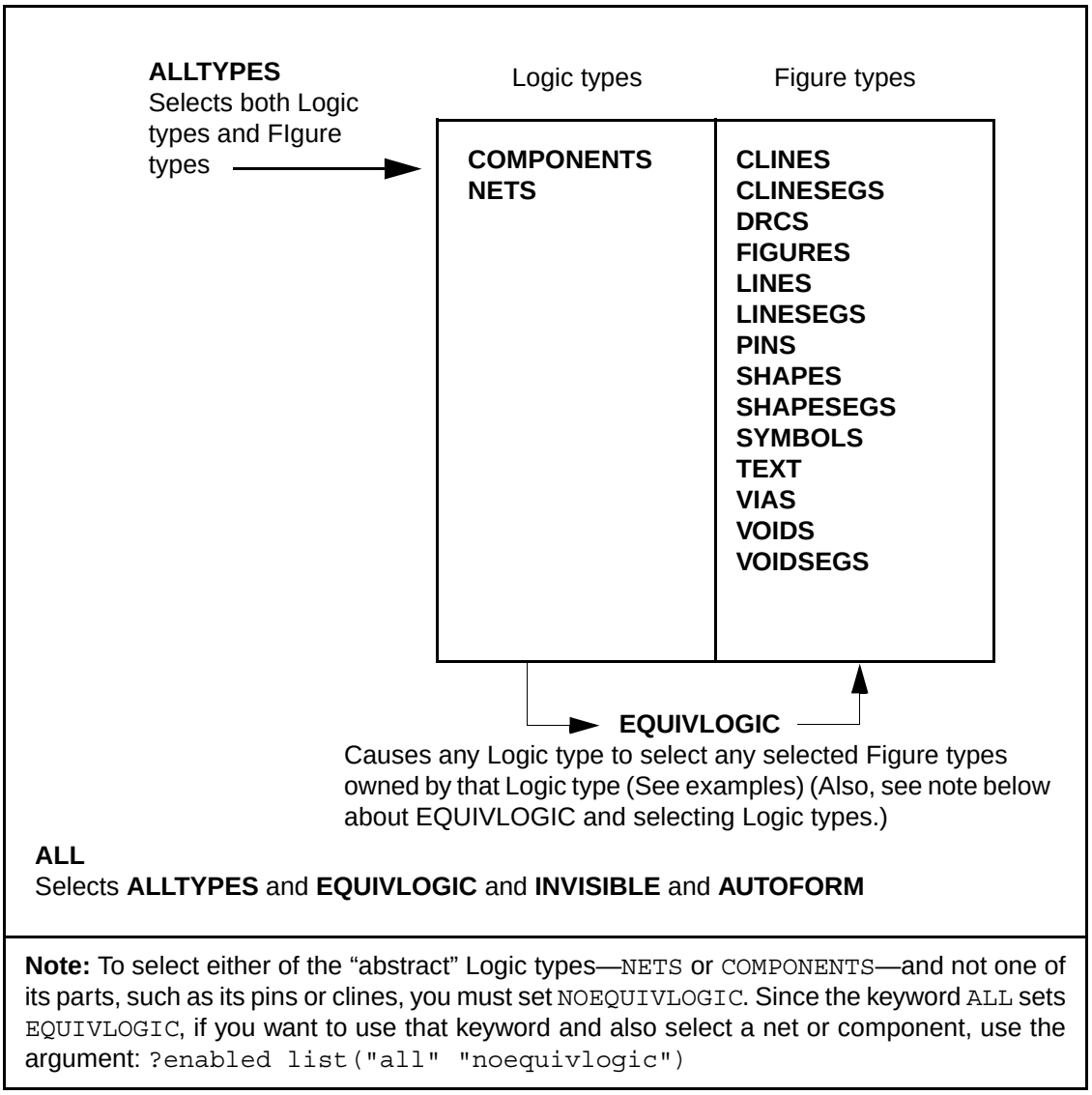
You need only pass `BOUNDARY_SHAPES` to the enabled list. It will be ignored if passed to the `onButtons` list.

Figure 4-1 Find Filter and Related Keywords



The differences in effects of different combinations of keywords can be subtle. Figure 4-2 on page 228 shows the relationship between the keywords.

Figure 4-2 Relationship Among axlSetFindFilter Keywords (*See examples also*)



Note: To select either of the “abstract” Logic types—`NETS` or `COMPONENTS`—and not one of its parts, such as its pins or clines, you must set `NOEQUIVLOGIC`. Since the keyword `ALL` sets `EQUIVLOGIC`, if you want to use that keyword and also select a net or component, use the argument: `?enabled list("all" "noequivlogic")`

Bond Objects

Bond objects (bond wires and fingers) are enabled only in SIP/APD. These options are ignored in the PCB products.

In SIP/APD, 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 seperately control the selection of bond objects (fingers or bond wires) from their base objects (vias or clines).

Note: You only need to make you Skill application bond smart if you wish to differentiate the disabling of one of these objects. By default, users in SIP/APD will be able to independantly select:

- VIAS or FINGERS if the Skill application enables VIAS
- CLINES and WIRES if CLINES are enabled.

Ideally most applications will not need to be updated to be bond smart.

Arguments

lt_enabled	List of keyword strings that describe object types that are to be selectable. Enabled object types will appear in the **Find Filter** form. Object types need the *onButton* 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.
lt_onButtons	List of keyword strings that describe object types that are to be enabled for selection. Enabled types will appear with the onButton depressed in the Find Filter form. onButton settings provide the default for controls which can be modified by the user when the Find Filter form is opened. An object type must be *on* in both the enabled and the onButton lists to be fully enabled for selection. Options are ignored when provided in the onButton list. Also supports a single keyword string instead of a list of strings.

Note: axlSetFindFilter does not display or select any types that are not enabled. That means that ?onButtons keywords only effect enabled types. For example, to have all enabled buttons be *on*, use ?onButtons list("alltypes"). In general, you need only set specific buttons *on* if you want those on and others off.

Value Returned

t	One or more Find Filter changes were made.

`nil`	No valid keywords were provided.

Application Programming Note

When you use `axlSetFindFilter` to implement an interactive command, make your AXL-SKILL program restore the user's FindFilter settings from the previous time he or she used the same command. Find Filter settings are incremental. Clear any previous settings as you start, then set the ones you want. Call `axlSetFindFilter` with `"noall"` as the first member of the list for both the `?enabled` and `?onButtons` arguments the first time you call it.

To maintain the user's Find Filter settings between invocations of your command

1. The first time the user invokes your program (when it loads), preset global variables to the list of `?enabled` and `?onButtons` settings you want as the initial default, as follows:

```
myglobal_enabled = list("noall" "xxx" "yyy" ... "zzz")
myglobal_onButtons = list("xxx" ..."zzz")
```

2. Each time the user invokes your command, call `axlSetFindFilter` with the current global values of `?enabled` and `?onButtons`.

```
axlGetFindfilter( ?enabled myglobal_enabled
     ?onButtons myglobal_onButtons)
```

   Since users can set or clear any of the buttons on the Find Filter, you need to save the button settings as you exit the command.

3. As you end the command, save the user's Find Filter *`onButton`* settings as shown:

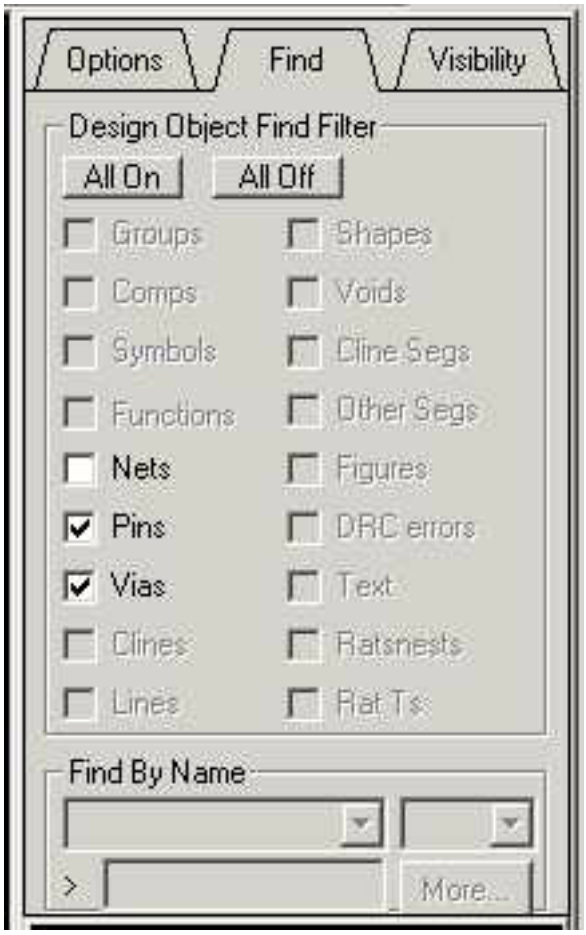
```
myglobal_onButtons = cons( "noall" axlGetFindFilter(t))
```

   Note: The `cons - "noall"` ensures that when you call `axlSetFindFilter` again you clear any settings left over from any previous command, and set only the buttons set at the time you call `axlGetFindFilter`.

Example 1

```
(axlSetFindFilter ?enabled (list "vias" "pins" "nets")
     ?onButtons (list "vias" "pins"))
```

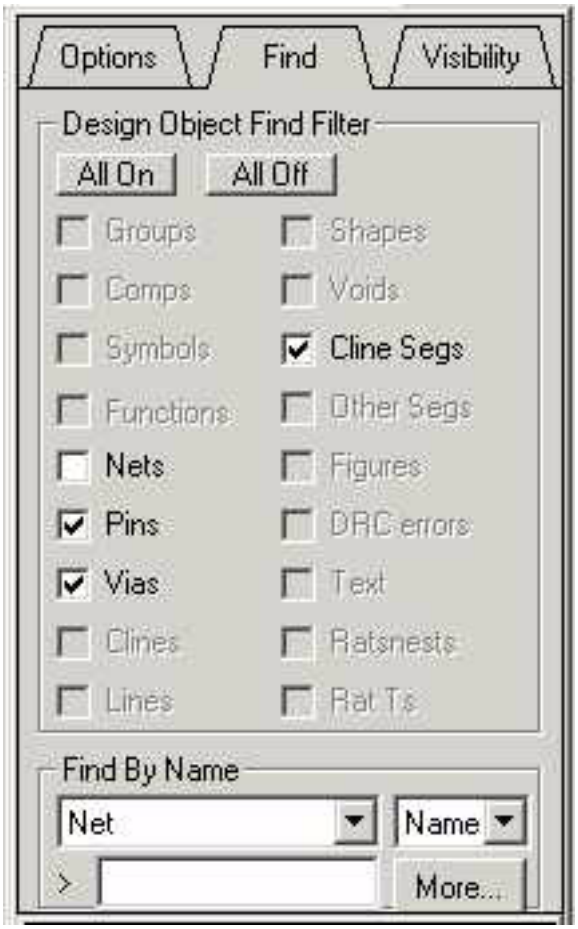
Displays the Find Filter with a list of Nets, Pins, and Vias. The Pins and Vias boxes are turned on as shown:



Example 2

```
(axlSetFindFilter ?enabled (list "noall" "vias" "pins"
     "nets" "clinesegs" "nameform")
     ?onButtons (list "vias" "pins" "clinesegs"))
```

Displays the Find Filter with Pins, Vias, and Clinesegs turned on.



Example 3

```
axlSetFindFilter(
?enabled list("noall" "equivlogic" "nets" "pins" "vias")
?onButtons list("all"))
```

Sets to find all the pins and vias on a net when the user selects any part of the net.

axlAutoOpenFindFilter

```
axlAutoOpenFindFilter(
)
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

`t`	Returns `t` always.

axlOpenFindFilter

```
axlOpenFindFilter(
)
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

`t`	Returns `t` always.

axlCloseFindFilter

```
axlCloseFindFilter(
)
⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

`t`	Returns `t` always.

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:

type	t_name	return type
'net	net name	`dbid` of a net
'refdes	refdes	`dbid` of a component instance
'padstack	padstack name	`dbid` 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")

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/lt_name
     [g_wildcard]
)
⇒ lo_dbid/nil
```

Description

Selects database objects by name.

Interface allows more than one name to be passed, but only one object type per call. For certain object types, a single name may return multiple objects. The supported object types and related items follow:

Object Type	Item the function finds
"NET"	net name
"COMPONENT"	component name
"FUNCTION"	function name
"DEVTYPE"	device type name
"SYMTYPE"	symbol type name
"PIN"	refdes.pinname
"REFDES"	find symbol by refdes name
"COMPREFDES"	find component by refdes name
"GROUP"	find group by name
"BUS"	find bus by name
"DIFF_PAIR"	find differential pair by name
"NETCLASS"	find netclass by name
"REGION"	find region by name
"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

Object Type	Item the function finds
"MATCH_GROUP"	find matchgroup 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

t_objectType	Type of database name.
t_name	Object to find.
lt_name	List of names to find.
[g_wildcard]	If * or ? appear in name, use regular expression matching

Value Returned

t	List of objects found.
nil	No matching name or illegal *objectType* name.

Example 1

```
Skill > p = axlSelectByName("NET" '("GND" "NET1"))
(dbid:28622576 dbid:28639844)
```

Finds two nets.

Example 2

```
Skill > p = axlSelectByName("NET" '("GND" "FOO"))
(dbid:28622576)
```

Finds two nets, but board only has GND.

Example 3

```
Skill > p = axlSelectByName("COMPONENT" "C1")
(dbid:28590388)
Skill > car(p)->objType
"component"
```

Finds Component C1.

Example 4

```
Skill > p = axlSelectByName("FUNCTION" "TF-326")
(dbid:28661572)
Skill > car(p)->objType
"function"
```

Finds function `TF-326.`

Example 5

```
Skill > p = axlSelectByName("DEVTYPE" "CAP1")
(dbid:28591032 dbid:28590700 dbid:28590388)
Skill > car(p)->objType
"component"
```

Finds devices of type `CAP1`.

Example 6

```
Skill > p = axlSelectByName("SYMTYPE" "dip14_3")
(dbid:28688416 dbid:28686192)
Skill > car(p)->objType
"symbol"
```

Finds symbols of type `DIP14_3`.

Example 7

```
Skill > p = axlSelectByName("PIN" "U1.1")
(dbid:28630692)
Skill > car(p)->objType
"pin"
```

Finds pin `U2.1`.

Example 8

```
Skill > p = axlSelectByName("REFDES" "U3")
(dbid:28688416)
Skill > car(p)->objType
"symbol"
```

Finds symbol by refdes `U3`.

Example 9

```
Skill > p = axlSelectByName("COMPREFDES" "U3")
(dbid:28621208)
Skill > car(p)->objType
"component"
```

Finds component by refdes `U3`.

Example 10

```
Skill > p = axlSelectByName("GROUP" "BAR")
(dbid:28593776)
Skill > car(p)->objType
```

Finds a group `BAR` by name.

Example 11

```
Skill > p = axlSelectByName("PIN" "U1.*" 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.

All names are case insensitive.

Note: Property names may change from release to release, or may be rendered obsolete. Skill programs using property names may require modifications in future releases.

Arguments

t_objectType	String for Allegro PCB Editor database object type. Must be: `compdef`, `component`, `drc`, `net`, `symdef`, `symbol`, or `group`.
t_property	String name of property.
t_value	Option property value.

g_regularExpression	t if property value is to be treated as a regular expression, or nil, which is the default, to treat property value as a simple match.

Value Returned

t	One or more *dbids* added to the select set.
nil	No *dbids* added to the select set.

Example 1

```
p = axlSelectByProperty("net" "ELECTRICAL_CONSTRAINT_SET")
axlAddSelectObject(p)
```

Selects all nets with an ECset property, then adds them to the current select set.

Example 2

```
p = axlSelectByProperty("net" "ELECTRICAL_CONSTRAINT_SET", "SPITFIRE_ADDRESS")
```

Selects all nets with ECset property of value SPITFIRE_ADDRESS.

Example 3

```
p = axlSelectByProperty("net" "ELECTRICAL_CONSTRAINT_SET", "SPITFIRE*" t)
```

Selects all nets with ECset property with any value matching spitfire.

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()
```

5

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.

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

Argument..	Value...
`bondWires`	`dbid` or list of `dbid`, representing the bond wires to be deleted.
`deleteFingers`	■ `t`: Bond fingers connected to the wires are removed ■ `nil`:Connect bond fingers are not deleted

Value Returned

Value is `t` returned, if one or more objects are deleted; otherwise the return value is `nil`.

axlChangeWidth

```
axlChangeWidth(
                  lo_dbid/o_dbid
                  f_newWidth
         )
         ==> lo_dbid/nil
```

Description

Changes width of lines, clines and segments (arc and line).

Note: If you need to change the width of multiple lines, it is more efficient to pass them as a list of *dbids* than to call this function for each *dbid*. This function does not support change in the width of shape borders.

Arguments

lo_dbid/o_dbid	Single dbid or list of dbids.
f_newWidth	New width of line.

Value Returned

List of width objects or nil if failed.

Failures:

- *dbid* is not a cline, line or line/arc segment of a line/cline.
- Illegal option types.
- Transformed object is outside of database extents.

Example

Changes the width of a cline to 20 in current database use units

```
; ashOne is a selection utility found at
;   <cdsroot>/pcb/examples/skill/ash-fxf/ashone.il
dbid = ashOne()
; pick a line, cline or segment (set find filter)
updatedDbid = axlChangeWidth(dbid, 20.0)
```

See Also

axlTransformObject

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 axlTransObject except it copies and transform's one or more objects.

One additional option is supported which is retainNet. This only applies to vias. If this option is t the net of the via is retained on copy, nil allows the via to connect to whatever it touches at the new location.

properties and attached text are also copied. Also ***see axlTransObject 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

`lo_dbid/o_dbid`	a single dbid or list of dbids
`allOrNone`	if t and a group of objects, transform must succeed on all objects or fail
`retainNet`	t/nil (applies to vias only)

Value Returned

list of transformed objects or nil if failed.

see

PERFORMANCE

If operating 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: axlTransObject, axlDBCloak

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). 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). 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.

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)
axlDBAltOrigin('origin sym)
; prints (3503.0 1058.0) -- origin of symbol is same as xy
axlDBAltOrigin('center sym)
; prints (3250.0 1737.0)
axlDBAltOrigin('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 charactisitics of a text string in the layout. 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 rename the reference designators and ignore the HARD_LOCATION property and property use axlRenameRefdes.

Arguments

`o_dbid`	Databased ID of text
`t_text`	Text string. If the value of this argument is set to `nil`, current settings are retained on the text
`r_textOrientation`	Orientation of text The `nil` value indicates that current settings are to be retained on the text. See Structure.
`x_textBlock`	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	defstruct not created
l_result	List containing: ■ (car) list of DBID of the text ■ (cadr) t if DRCs created or nil.

Do not pass text string with newlines as an argument.

See Also: axlTransformObject, 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)
```

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 ripup 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)
subclasses - subclasses must by 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

o_dbid/lo_dbid	*dbid*, or list of *dbids* to delete from layout.
g_mode	optional delete options. 'ripup - enable etch ripup option (same as Allegro delete ripup command ripup option)

Value Returned

`t`	Deleted one or more objects from the layout.
`nil`	Deleted no objects from the layout.

If passed component or net dbid will delete the logic. This is different from the Allegro `delete` ***command which will delete the physical objects associated with the logic (clines/vias for nets and 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()))
          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"))
```

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

lo_attach	List of *dbids* of objects from which properties are to be deleted. *lo_attach* may be a single *dbid*. If *lo_attach* is `nil`, then the property is to be deleted from the design itself.
lt_name	List of names of the properties to be deleted. *lt_name* may be a list of strings for several properties, or a single string, if only one property is to be deleted.

Value Returned

l_result	List. (`car`) list of *dbids* of members of *lo_attach* that successfully had at least one property deleted. (`cadr`) always `nil`.
`nil`	No properties deleted.

See Also

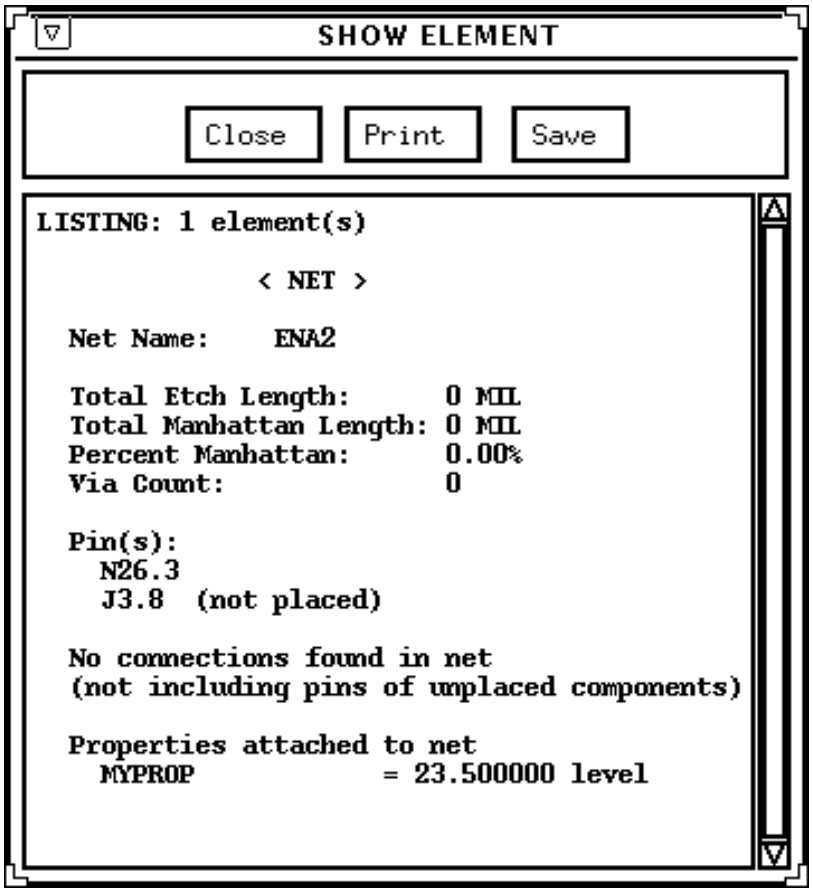
axlDBAddProp, axlDBDeletePropAll

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.

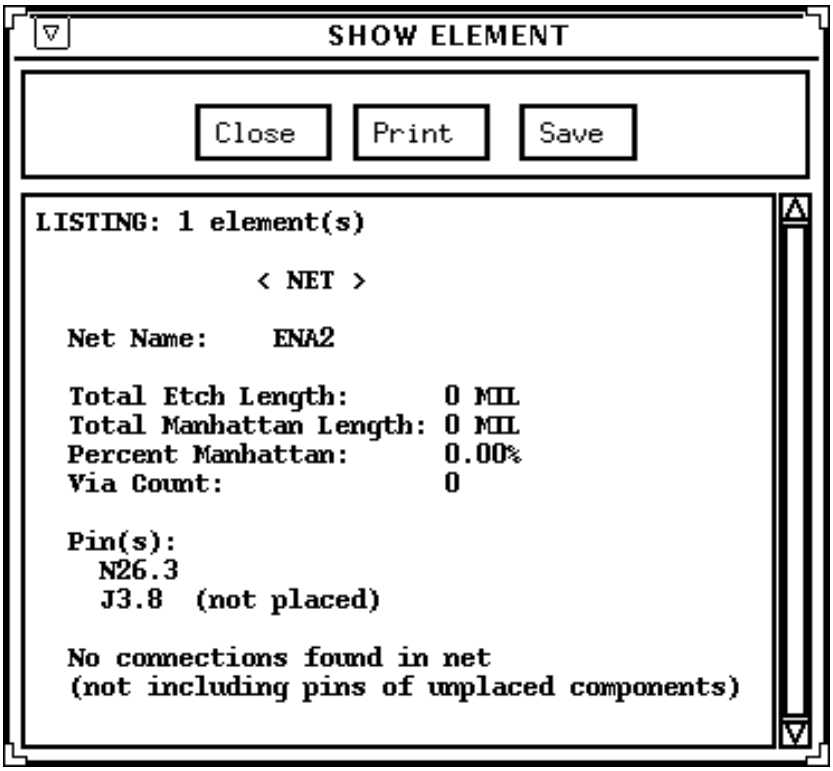
The following **Show Element** form shows the net with `"MYPROP"` attached.



```
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

x_count	Returns number of properties deleted
nil	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")
```

axlDBOpenShape

```
axlDBOpenShape(
             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

axlDBCloseShape

axlDBCreateOpenShape, axlDBCreateVoid, axlShapeDeleteVoids, axlShapeAutoVoid

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)
```

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, etc. 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 viewable by the user.

Arguments

None.

Value Returned

l_bBox	bBox of the 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.

Changing the padstack on a pin in the drawing editor results in an exploded pin which increases your database size and impacts refresh_symbol.

Using this function can result in disconnects and new DRC violations.

Performance Hints

To change all instances of a particular padstack, it is faster to change the padstack itself.

If you are changing many pins and vias to the same padstack, you can save time by calling this function with a list of pins/vias instead of calling it for each pin or via.

axlDeleteFillet

```
axlDeleteFillet(
     o_dbid
)
⇒ t/nil
```

Description

Deletes fillet 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
)
⇒ t/nil
```

Description

Adds fillet between cline and pin/via, and at T. Removes and re-generates existing fillets. Fillet parameters are controlled from the Glossing **Pad and T Parameter** form.

Arguments

o_dbid	*dbid* can either be a NET or CLINE.

Value Returned

t	Fillet added.
nil	No fillet added.

Notes

Pins, vias and Ts are not supported; use axlDBGetConnect on these objects to get a list of clines that connect.

For best performance, especially if fillets impact dynamic shapes, make a single call with the list of objects to be filleted.

Examples

```
fillet new MEMDATA8

axlFillet(car(axlSelectByName("NET" "MEM_DATA8")))
```

axlPurgePadstacks

```
axlPurgePadstack (
     S_mode
     t/nil

)

⇒ x-cnt
```

Description

Purges unused padstacks from the database in the area controlled by *S_mode* symbol.

‘

S_mode symbol	2nd arg = `t`	2nd arg = `nil`
`'padstacks`	Only purges unused derived padstacks.	Purges all unused padstacks.
`'via`	Purges vias not found from all via list constraints under the physical rule set and purges vias not loaded in the database, but found by looking on the disk via the `PSMPATH` environment variable.	Purges vias not found from all the via list constraints under the physical rule set. The `nil` option is NOT available from the Allegro PCB Editor user interface.

Tip

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.

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/axlcore/ashshape.il`

```
axlShapeAutoVoid(shapeDbid '(noRipThermals fragment))
```

axlShapeChangeDynamicType

```
axlShapeChangeDynamicType(
                  o_shapeId
                  g_dynamic
                  g_msgs
          ) -> o_dynShapeId/l_staticShapeId/nil
```

Description

Swaps a connectivity shape from static to dynamic or the reverse. This offers the same functionality as the 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.

Tip

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

o_shapeId	Dynamic shape id or static id.
g_dynamic	`t` makes the shape dynamic, `nil` makes the shape static.
g_msgs	`t` issue error messages if failed to convert; else be silent

Value Returned

`nil`	Failure.
o_dynShapeId	`dbid` of the dynamic shape converted from static.
l_staticShapeId	List of static shapes converted from dynamic shapes.

See Also

axlShapeChangeDynamicType

Examples

See <cdsroot>/share/pcb/examples/skill/axlcore/ashshape.il

Change to dynamic shape with messages:

```
ret = axlShapeChangeDynamicType(shape t t)
```

Change to static shape; no messages

```
ret = axlShapeChangeDynamicType(shape nil nil)
```

axlShapeDeleteVoids

```
axlShapeAutoVoid(
            o_shapeId/o_voidId/lo_voidid
         ) -> t/nil
```

Description

Lets you delete voids in a shape. Supports the following forms of arguments:

- Shape 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 axlDBOpenShape and axlDBCreateCloseShape.

Arguments

o_shapeId	Given a shape; deletes all voids associated with that shape.
o_voidId	Deletes the given void.
lo_voidid	Deletes the list of voids.

Value Returned

t	Deletes voids.
nil	Error.

See Also

axlShapeAutoVoid, axlDBOpenShape, axlDBCreateCloseShape

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

`o_shapeDbid`	`dbid` a dynamic shape.
`g_force`	Force shape to update even if it is up to date.

Value Returned

`x_ood`	If updating all returns count of all shapes that failed in updating. If single shape returns `0`; update successful, `1` otherwise.
`nil`	Return if there is an error; `dbid` is not a dynamic shape.

Examples

Force update of one dynamic shape:

```
axlShapeDynamicUpdate(shapeId, t) -> 0
```

Update all shapes `ood`:

```
axlShapeDynamicUpdate(nil nil) -> 0
```

axlShapeRaisePriority

```
axlShapeRaisePriority(
                  o_shapeId
         ) -> x_priority/nil
```

Description

Raises the voiding priority of a dynamic shape (*o_shapeId*) to the highest on the chosen layer. If this shape overlaps other dynamic shapes on the layer, the other shapes void away from this shape.

The priority number is relative. 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

x_priority > 0	New priority of shape.
`-1`	Already at highest priority.
`nil`	Not a dynamic shape.

See Also

`axlShapeChangeDynamicType`

Example

See <cdsroot>/share/pcb/examples/skill/axlcore/ashshape.il

```
axlShapeRaisePriority(shape)
```

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).

Tip

If changing type of multiple shapes or doing multiple operations on a single shape (e.g. convert then raise priority) consider wrapping the code in axlDBCloak to batch updates.

Arguments

o_shapeId - dynamic shape id or static id.

g_dynamic - t make shape dynamic, nil make static

g_options - Available options are

'check - don't merge only perform checks for merging

'quiet - don't 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

See `<cdsroot>/share/pcb/examples/skill/axlcore/ashshape.il`

- Change to dynamic with messages

```
ret = axlShapeChangeDynamicType(shape t t)
```

- Change to static no messages

```
ret = axlShapeChangeDynamicType(shape nil nil)
```

axlShoveItems

```
list

axlShoveItems(
     l_itemList

)

⇒ t/nil
```

Description

Takes a list of *dbids* and shoves them according to the parameters set using axlShoveSetParams.

Arguments

l_itemList	List of *dbids* (clines, pins, or vias) to be shoved.

Value Returned

t	One or more items shoved.
nil	No items shoved.

Note: Pins and vias are not shoved, but the clines around them are shoved in an attempt to eliminate any DRCs between the pin/via and the cline.

The list of *dbids* passed in does not reflect the results of the shove, as the original item may be deleted and/or replaced.

Example

```
(defun ShoveElement ()
     axlSetFindFilter (?enabled '("CLINES" "VIAS")
                      ?onButtons '("CLINES" "VIAS"))

     axlSelect()
     axlShoveItems (axlGetSelSet())
)
```

Shoves an item (or items) interactively selected by the user.

axlShoveSetParams

```
axlShoveSetParams(
     l_params
)
⇒ t/nil
```

Description

Sets the parameters used for shoving by the `axlShoveItems`. If you do not provide all values, the indicated default is used.

Arguments

`l_params` List of parameters of the form:

```
(shoveMode cornerType gridded smooth oop samenet)
```

`ShoveMode` is an integer as shown:

shoveMode	Description
`0`	hug preferred - Items passed in try to mold around items they are in violation with (default)
`1`	shove preferred - Items passed in try to shove items they are in violation with.

`CornerType` is an integer as shown:

cornerType	Description
`90`	90 degree corners.
`45`	45 degree corners.
`0`	Any angle corners.

Gridded is an integer as shown:

gridded	Description
0	Ignore grids (default)
1	Perform shoves on grid.

Smooth allows smoothing of shoved traces and is an integer as shown:

smooth	Description
0	No smoothing (default)
1	Minimal smoothing.
2	More smoothing.
3	Still more smoothing.
4	Full smoothing.

Oops allows aborting the shove of DRCs result and is an integer as shown:

oops	Description
0	*Oops* off (default)
1	*Oops* if drcs are left over.

Samenet tests for samenet violations.

Note: This results in a post-shove check for drcs that is meaningful only if you also set *oops* to the "oops if drcs" value.

samenet	Description
0	No *samenet* tests (default).
1	Enable *samenet* 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 diffpair aware.
- may have issues with electrically constrained nets

Tip

See axlDBIgnoreFixed if you want to temporary disable FIXED testing.

Arguments

`l_params` List containing the parameters, the list is of the folowing format.

```
(cornerType maxCornerLength padEntryRestriction minPadEntryLength
     sortDirection)
```

`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 dbunits. 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 params

```
axlSmoothSetParams(list("45" -1.0 "0" 10.0 0))
```

- Update cornertyp to 90

```
axlSmoothSetParams(list("90" nil nil nil nil))
```

See Also

axlSmoothItems, axlSmoothDesign, axlDBIgnoreFixed

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
     ?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). If the object does not accept a transform, then that transform is silently ignored.

If multiple transformations are applied, the order used is:

1. move
2. mirror
3. rotate

If `allOrNone` flag is set, then the entire transformation fails when one object's transformation fails. By default, one object's failure does not stop the transformation on the other objects. A failure is a database failure. For example, a move that puts an object outside of the database extents is a database failure. Attempting an illegal transform is NOT a failure. If one or more objects are not transformed, there is no failure.

OBJECT	MOVE	MIRROR	ROTATE	SPIN	ORIGIN (6)	NOTES
segments	X	X	X		box	
cline	X	X	X		box	
line	X	X	X		box	
symbol	X	X	X		xy	
shape	X	X	X		box	
text	X	X	X		xy	
pin	X		X		xy	4, 5
via	X	X	X		xy	
rat_t	X		X		xy	
group	X	X	X		xy	8

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. Rotation (angle option) works as follows:
   - Positive angle results in a counter-clockwise rotation.

- If just angle is provided, then the object is rotated about its origin point. If the *dbid* has no origin, then the center of its bounding box is used. If a list of *dbids* is provided, then the rotation always occurs about the center of the object set.
- You can provide a rotation origin.

  (?origin *l_rotatePoint*).

  This point is then used as the rotation point.

Cautions

- 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

lo_dbid/*o_dbid*	Single *dbid* or a list of *dbids*.
l_deltaPoint	Move distance.
mirror	Mirror object (see table)
f_angle	Rotation angle.
l_rotatePoint	Rotation point.
allOrNone	If t and a group of objects, transform must succeed on all objects, or fail.

Value Returned

lo_dbid	List of transformed objects.
nil	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

axlTransObject, axlDBCloak

Examples

`dbid` represents one database objects.

`ldbid` represents a list of database objects.

Example 1

```
axlTransformObject(ldbid, ?move '(100.0, 0.0))
```

Moves a set of objects `100` database units vertically.

Example 2

```
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.

6

Database Read Functions

AXL-SKILL Database Read Functions

The chapter describes the AXL-SKILL functions that read the Allegro PCB Editor database.

axlDBGetDesign

```
axlDBGetDesign(

)

⇒ o_design/nil
```

Description

Returns the root design *dbid*. Use this *dbid* to get the design properties and to add properties to the design.

Note: Note that you cannot edit the root design object. AXL-SKILL edit commands ignore this *dbid*.

Arguments

None.

Value Returned

o_design	Root design *dbid*.
nil	Error occurred.

Example

```
mydesign = axlDBGetDesign()
axlDBAddProp( mydesign, list("board_thickness", 0.350))
```

Gets the root design and sets the *BOARD_THICKNESS* property to 0.350 inches.

To verify the property has the value specified:

1. From the 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.

axlGetDieType

```
axlGetDieType(
     componentDBID
)
==> t_dieType
```

Description

Returns the die attachment type for a given die component in a Cadence packaging tool (APD/SIP). A die is considered to be an IC class component in the database. Currently, the supported attachement types include the following:

- WIREBOND
- FLIP CHIP

Arguments

Component DBID	*dbid* handle of the die component to query.

Value Returned

t_dieType	If successful.
nil	If failed (non-die object passed or not a packaging product).

Examples

```
axlGetDieType(myComp)
     ==> "FLIP CHIP"
```

axlDBGetDrillPlating

```
axlDBGetDrillPlating
     t_padstackname
)

⇒ "PLATED"/"NON PLATED"/"OPTIONAL"/nil
```

Description

Retrieves the plating type of the padstack passed as an argument to this function.

Arguments

t_padstackname	Name of padstack.

Value Returned

Plated/Nonplated/Optional	Drillplating name.
nil	Incorrect padstack name, or other error occurred.

axlIsDBIDType

```
axlIsDBIDType(
     g_dbid
)
⇒ t/nil
```

Description

Determines if *g_dbid* is an 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:

```
"This is NOT an Allegro DBID."
"This is an Allegro DBID."
```

axlDBGetAttachedText

```
axlDBGetAttachedText(
     o_dbid
)
⇒ l_dbid/nil
```

Description

Returns the list of *dbids* of text objects attached to the object whose *dbid* is *o_dbid*.

Arguments

o_dbid	*dbid* of object from which attached text *dbids* are retrieved.

Value Returned

l_dbid	List of the text objects attached to *o_dbid*.
nil	No attached text objects.

Example

```
(defun showText ()
     "Print text of selected objects"
     mypopup = axlUIPopupDefine( nil
          (list (list "Done" 'axlFinishEnterFun)
          (list "Cancel" 'axlCancelEnterFun)))
axlUIPopupSet( mypopup)
axlSetFindFilter( ?enabled list("noall")
     ?onButtons "noall")
axlSetFindFilter( ?enabled list("symbols")
     ?onButtons "symbols")
axlOpenFindFilter()
(while (axlSelect)
     progn(
     alltext =
          axlDBGetAttachedText(car(axlGetSelSet()))
          foreach(thistext alltext
               printf( "Text on this symbol is : '%s'\n",
               thistext->text))))
axlCloseFindFilter())
```

Lets the user pick a symbol, then prints the text attributes of each text object attached to that symbol.

Run `showText()` and pick a symbol of device type `"74F74"`, assigned as `refdes "T23"`. The function prints the following:

```
Text on this symbol is : 'T23'
Text on this symbol is : '74F74'
```

axlDBGetPad

```
axlDBGetPad(
     o_dbid
     t_layer
     t_type
)
⇒ o_pad/nil
```

Description

For the pin or via specified by *o_dbid*, gets the pad of type *t_type* associated with layer *t_layer*.

Arguments

o_dbid	*dbid* of the pin, via, or a padstack definition.
t_layer	Layer of pad to retrieve, for example, `"ETCH/TOP"`.
t_type	Type of pad to retrieve: `"REGULAR"`, `"ANTI"`, or `"THERMAL"`.

Value Returned

o_pad	*dbid* of the pad of the type associated with *o_dbid* on the layer specified.
`nil`	Cannot get the pad *dbid*.

Example

```
(defun showPad ()
     mypopup = axlUIPopupDefine( nil
          (list (list "Done" 'axlFinishEnterFun)
          (list "Cancel" 'axlCancelEnterFun)))
axlUIPopupSet( mypopup)
axlSetFindFilter( ?enabled list("noall")
     ?onButtons "noall")
axlSetFindFilter( ?enabled list("pins" "vias")
          ?onButtons list("pins" "vias"))
(while axlSelect()
     progn(
          mypad = axlDBGetPad(car(axlGetSelSet())
               "etch/top" "regular")
          printf( "Pad figure type : %s\n",
               mypad->figureName))))
```

Lets the user pick any pin or via and shows the *figureName* attribute of the selected pad.

Run `showPad()` and pick a pin with a square pad on `"etch/top"`, then a circular pad. The function prints the following:

```
Pad figure type : SQUARE
Pad figure type : CIRCLE
```

axlDBGetPropDictEntry

```
axlDBGetPropDictEntry(
     t_name
)
⇒ o_propDictEntry/nil
```

Description

Gets the property dictionary entry for the property name given by the string *t_name*. . If name is `nil` 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*.
lt_validObjects:	List of valid objects to associate with a property
`nil`	Could not get the entry.

See Also

axlDBAddProp

Example

The following example gets the `"signal model"` property.

```
myprop = axlDBGetPropDictEntry("signal_model")
```

and dumps its attributes

```
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

o_dbid	*dbid* of the object from which to get the properties.
lt_type	List of strings qualifying the types of properties to be retrieved from *o_dbid*. `"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

l_result	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"))
```

axlDBGetDesignUnits

```
axlDBGetDesignUnits(
)
⇒ l_value/nil
```

Description

Returns the design units and accuracy number of the active design.

Arguments

None.

Value Returned

l_value	List containing the design units as a string and the accuracy number as an integer.
nil	Failed to return the design units and accuracy number of the active design.

Example

```
(axlDBGetDesignUnits)
     ⇒("millimeters" 3)
```

The design **Drawing Parameters** form shows *User Units* as `Millimeter` and *Accuracy* as `3`.

axlDBRefreshId

```
axlDBRefreshId(
     o_dbid/nil
)
⇒ o_dbid/nil
```

Description

Updates the attributes of the object specified by *o_dbid*. Subsequent attribute retrieval requests access the updated information.

Note: Because of performance considerations, refreshes only the object itself. If the object being refreshed has *dbids* in any of its attributes, those *dbids* are not refreshed. For example, a net branch has *children*, a list of paths, tees, vias, pins, and shapes. If another path is added to that list of paths due to connectivity change, `axlDBRefreshId` of the branch does not update the *children*. If you move a via that is a child of the branch, then doing `axlDBRefreshId` 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

o_dbid	Refreshed *dbid*.
`nil`	Could not refresh.

Example

```
axlSetFindFilter( ?enabled
     '("noall" "alltypes"))
axlSingleSelectName("net" "sclkl")
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* `"sclkl"`, 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

l_dbid	List of standalone branches.
`nil`	No standalone branches found.

Example

```
(axlDBGetLonelyBranches)
⇒(dbid:12051156 dbid:11994768 dbid:12002292 dbid:12000892 dbid:11999396
dbid:11996652 dbid:11996048 dbid:11994476 dbid:11992964 dbid:11991564
dbid:11989672 dbid:11989344 dbid:12072172 dbid:11895392 dbid:11892048
dbid:11888704 dbid:11888744 dbid:11888804 dbid:11888844 dbid:11888884
dbid:12074948 dbid:11888984 dbid:11889064 dbid:11889204 dbid:11889224
dbid:11889856 dbid:11890036 dbid:11890056 dbid:11890236 dbid:11890256
dbid:11886180 dbid:12011360 dbid:11886760 dbid:11887140 dbid:11887916 )
```

Gets list of standalone branch *dbids*.

axlDBGetConnect

```
axlDBGetConnect(
     o_dbid
     t_full

)

⇒ l_result/nil
```

Description

Finds all the elements, including pads and shapes, that are connected to a given *dbid*. Input can be a PIN, VIA, T, CLINE/CARC or CLINE/CARC SEGMENT.

If *t_full* is `nil`, the function returns a list of objects connected to either end for CLINES and segments. The function returns a list of connected clines for pins, vias, or Ts.

If *t_full* is `t`, the function returns connectivity of the *dbid* including shapes.

Note: You should set *t_fill* to `t`. The `nil` option is only available for legacy purposes.

Arguments

o_dbid	A *dbid*, cline, 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*.

axlDBIsBondpad

```
axlDBIsBondpad(
     o_dbid
)
⇒ t/nil
```

Description

Verifies whether or not the given element is a *bondpad*.

A *bondpad* (or *bondfinger*) is a `"via"` with the `BOND_PAD` property.

Arguments

o_dbid	*dbid* of the element to be checked.

Value Returned

`t`	*o_dbid* is a bondpad.
`nil`	*o_dbid* is not a bondpad.

axlDBIsBondwire

```
axlDBIsBondwire(
     o_dbid
)
⇒ t/nil
```

Description

Verifies whether or not the given element is a *bonding wire*.

A *bonding wire* (or *bondingfinger*) is a *"via"* (can be through hole or blind/buried) with either the *"beginning"* or *"ending"* (if the via has been mirrored) layer type set to the `BondingWire` property.

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.

axlDBIsDiePad

```
axlDBIsDiePad(
     rd_dbid
)
⇒ t/nil
```

Description

Verifies whether or not the given element is a *die pad*.

A *die pad* is a pin with a component class of `IC`.

Arguments

rd_dbid	*dbid* of the element to check.

Value Returned

`t`	*rd_dbid* is a die pad.
`nil`	*rd_dbid* is not a die pad.

axlDBIsFixed

```
axlDBIsFixed(
     o_dbid
     [g_showMessage]
)
⇒ nil or [dbid of 1st element that makes the item fixed]
```

Description

Verifies whether or not the database object is fixed.

An object can be fixed by the following:

- Object has the `FIXED` property or has parents with the `FIXED` property.
- Object is a symbol with test points and the `FIXED` test point flag is set.
- Object is a symbol and has one or more children with the `FIXED` property.

Returns the first item found that caused the element to be fixed.

Arguments

o_dbid	*dbid* of the element to check.
g_showMessage	Use `t` to have Allegro PCB Editor display the message if the item is fixed or `nil` to have no message display.

Value Returned

dbid	*dbid* of the element causing the object to be fixed.
`nil`	Object not fixed.

Example

```
p = axlSelectByname("SYMBOL" "U1")
ret = axlDBIsFixed(p)
```

axlDBIsPackagePin

```
axlDBIsPackagePin(
     rd_dbid
)
⇒ t/nil
```

Description

Verifies whether or not the given element is a *package pin*.

A *package pin* is a pin with a component class of IO.

Arguments

rd_dbid	*dbid* of element to check.

Value Returned

t	*rd_dbid* is a package pin.
nil	*rd_dbid* is not a package pin.

axlDBIsPlatingbarPin

```
axlDBIsPlatingbarPin(
     rd_dbid
)
⇒ t/nil
```

Description

Verifies whether or not the given element is a *plating bar pin*.

A *plating bar pin* is a pin with a component class of `DISCRETE` or `PLATING_BAR`.

Arguments

rd_dbid	*dbid* of the element to check.

Value Returned

`t`	*rd_dbid* is a plating bar pin.
`nil`	*rd_dbid* is not a plating bar pin.

axlGetModuleInstanceDefinition

```
axlGetModuleInstanceDefinition(
     o_modinst
)
⇒ t_moddef/nil
```

Description

AXL interface to the C function that returns the name of the module definition used to create the module instance.

Arguments

o_modinst	AXL *dbid* of the module instance (the *dbid* returned by axlDBCreateModuleInstance.)

Value Returned

t_moddef	String containing the name of the module definition.
nil	Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))
axlSingleSelectName("GROUP" "inst")
modinst = car(axlGetSelSet())
axlGetModuleInstanceDefinition(modinst)
= "mod"
```

Gets the definition of a module instance named inst.

axlGetModuleInstanceLocation

```
axlGetModuleInstanceLocation(
     o_modinst
)
⇒ l_loc/nil
```

Description

AXL interface to the C function that gets the current location of the module instance in the design.

Arguments

o_modinst	AXL *dbid* of the module instance (the *dbid* returned by `axlDBCreateModuleInstance`.)

Value Returned

l_loc	List of data describing the location of the module instance. The list syntax is as follows. `list(l_origin, x_rotation [g_mirror])` where ■ `l_origin:` origin of module ■ `x_rotation:` rotation in degress * 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

o_modinst	AXL *dbid* of the module instance (the *dbid* returned by axlDBCreateModuleInstance.)

Value Returned

i_logic	Value of the logic method flag for the module instance. Legal values are: 0 - no logic 1 - logic from schematic 2 - logic from module definition
nil	Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))
axlSingleSelectName("GROUP" "inst")
modinst = car(axlGetSelSet())
axlGetModuleInstanceLogicMethod(modinst)
= 2
```

Gets the logic method of a module instance named inst.

axlGetModuleInstanceNetExceptions

```
axlGetModuleInstanceNetExceptions(
     o_modinst
)
⇒ l_nets/nil
```

Description

AXL interface to the C function that gets the net exception of the module instance in the design.

Arguments

o_modinst	AXL *dbid* of the module instance (the *dbid* returned by axlDBCreateModuleInstance.)

Value Returned

l_nets	List of names of the nets that are treated as exceptions in the module instance.
nil	Could not access the information.

Example

```
axlSetFindFilter(?enabled '("noall" "groups") ?onButtons '("noall" "groups" ))
axlSingleSelectName("GROUP" "inst")
modinst = car(axlGetSelSet())
axlGetModuleInstanceNetExceptions(modinst)
= ("GND" "+5")
```

Gets the list of net exceptions of a module instance named inst.

axlIsDummyNet

```
axlIsDummyNet(
     net_dbid
)
⇒ t/nil
```

Description

Determines if a given net is a Dummy net.

Arguments

net_dbid	Net database object.

Value Returned

t	*net_dbid* is a Dummy Net.
nil	*net_dbid* is not a Dummy Net.

axlIsLayerNegative

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

Description

Determines whether or not the given plane layer is negative.

Arguments

t_layerName	Name of the conductor layer to check.

Value Returned

t	Active layer is negative.
nil	Active layer is not negative or is not an ETCH layer.

axlIsPinUnused

```
axlIsPinUnused(
     pin_dbid
)
⇒ t/nil
```

Description

Determines if a given pin is unused.

Arguments

pin_dbid	Pin database object.

Value Returned

t	Pin is unused.
nil	Pin is used.

axlIsitFill

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

Description

Determines if fill shape is allowed for a given class subclass.

Arguments

t_layer	Layer name, for example, ETCH/TOP.

Value Returned

t	Fill shape is allowed.
nil	Fill shape is not allowed.

axlOK2Void

```
axlOK2Void(
     t_layer
)
⇒ t/nil
```

Description

Determines if voids are allowed for a given *class/subclass*.

Arguments

t_layer	Layer name, for example, ETCH/TOP.

Value Returned

t	Voids are allowed.
nil	Voids are not allowed.

axlDBAssignNet

```
axlDBAssignNet(
     o_object/lo_object
     o_net/t_net
     [g_ripup]
)
⇒ t/nil
```

Description

Assigns an object or a list of objects to a new net. Currently supports pins, vias, and shapes. Vias may not stay on net assigned if they do not connect to an object on the new net.

Arguments

o_object	*dbid*, or list of *dbids* of objects to change net.
o_net	*dbid* of destination net, or `nil` if assigning to a dummy net.
t_net	Net name.
g_ripup	`t` = ripup clines connected to modified objects. `nil` = do not ripup clines connected to modified objects. The default is `nil`.
g_ignoreFixed	By default, won't allow net assignment if FIXED property is present on the object in question. Also clines with the FIXED property if they connect to the object won't move to the new net of the object. Setting this argument to *t* ignores the FIXED property.

Value Returned

`t`	At least one object changed net.
`nil`	No object changed net.

See Also

axlDBCreateNet

Example

```
pin_id->net->name
     => ""
net_id->name
     => "GND"
axlDBAssignNet(pin_id net_id t)
     => t
pin_id->net->name
     => "GND"
```

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

None if not created, or a axl `dbid` of net

See Also

axlDBAssignNet

Example

```
net = axlDBNetCreate("gnd")
=> dbid:123456
net->name

"GND"
```

axlDBDynamicShapes

```
axlDBDynamicShapes(
     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

g_value	`t` = update dynamic shapes `nil` = return count of out of date shapes

Value Returned

x_count	Count of out of date shapes. If updating shapes, *x_count* is the number of out of date shapes before the update.

axlDBGetShapes

```
axlDBGetShapes(
     t_layer
)
⇒ l_dbid/nil
```

Description

Provides quick access to shapes without access to visibility or find settings.

Arguments

t_layer	Layer name nil = all layers <*class*> = all subclasses of the class <*class*>/<*subclass*> = specified layer

Value Returned

l_dbid	List of shapes.
nil	Incorrect argument.

Example 1

```
axlDBGetShapes(nil)
```

Returns all shapes on the design.

Example 2

```
axlDBGetShapes("BOUNDARY")
```

Returns all shapes on the BOUNDARY layer.

Example 3

```
axlDBGetShapes("ETCH/GND")
```

Returns all shapes on ETCH GND.

Example 4

```
axlDBGetShapes("ROUTE KEEPOUT")
```

Returns all shapes on ROUTE KEEPOUT.

axlDBIsBondingWireLayer

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

Description

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.

Verifies if a layer is a bonding layer. This means that attribute of the “paramLayer” parameter *dbid* called “type” has a value of “`BONDING _WIRE`”.

This is normally used in the APD product.

Arguments

t_layerName	Layer name, for example, "CONDUCTOR/<*subclass*>" **Note:** CONDUCTOR is ETCH in Allegro PCB Editor.

Value Returned

`t`	Layer is a bonding layer.
`nil`	Layer is not a bonding layer.

See Also

`axlDBIsBondwire`

Example

```
axlDBIsBondingWireLayer("CONDUCTOR/TOP_COND")-> nil
```

axlDBTextBlockCompact

```
axlDBTextBlockCompact(
     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 = axlDBTextBlockCompact(nil)
printf("This database has %d unused text blocks\n" unused)
```

7

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 14, “Database Create Functions,” and the `axlAddSimpleXXXDynamics` functions described in this chapter.

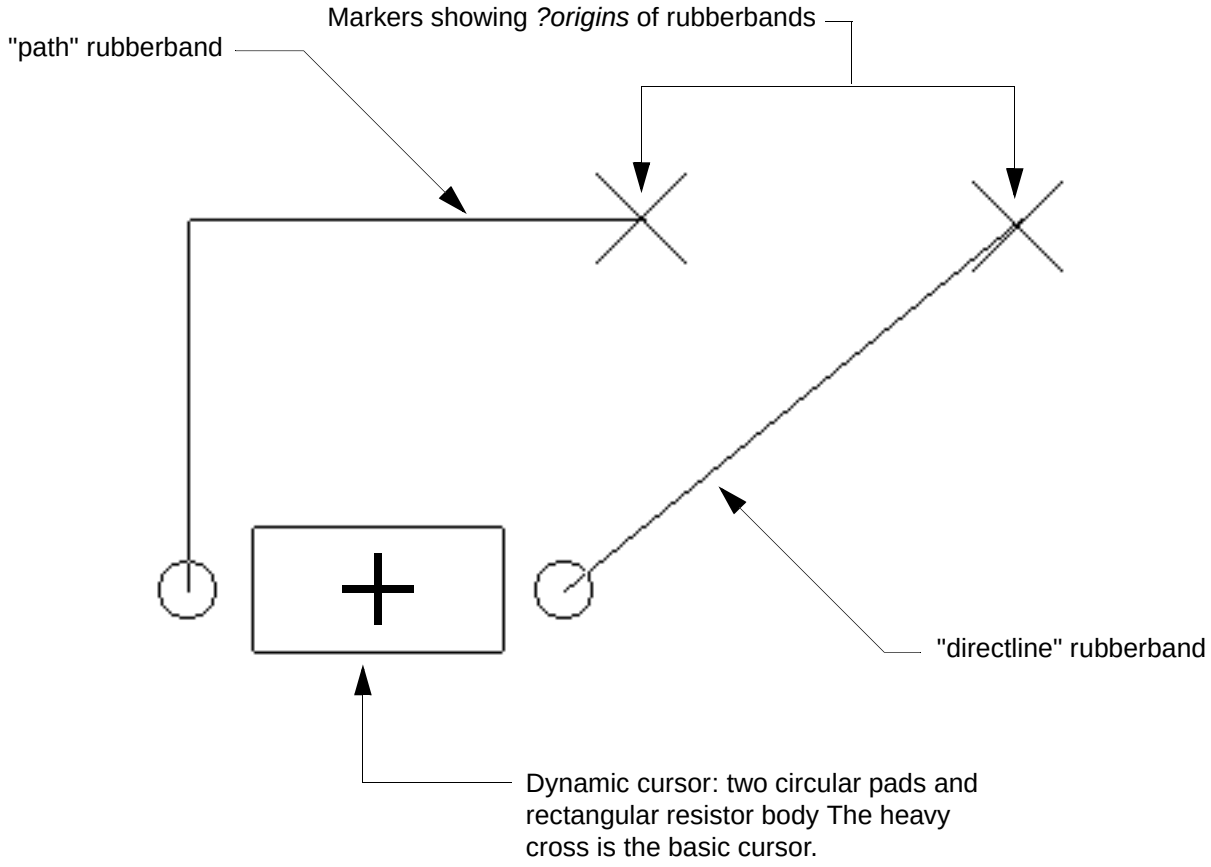
Example 1: Dynamic Rubberband

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()
; Create cross markers to show rubberband origins:
axlDBCreateLine(list(9150:4450 9050:4550) 0.
                 "board geometry/dimension")
axlDBCreateLine(list(9150:4550 9050:4450) 0.
                 "board geometry/dimension")
axlDBCreateLine(list(8550:4450 8450:4550) 0.
                 "board geometry/dimension")
axlDBCreateLine(list(8550:4550 8450:4450) 0.
                 "board geometry/dimension")
mypath = axlPathStart(list( -350:0)) ; Start circular pad
axlPathArcCenter(mypath, 0., -350:0, nil, -300:0)
; Load the first pad into the dynamic cursor buffer
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)
mypath = axlPathStart(list( 350:0)) ; Start circular pad
axlPathArcCenter(mypath, 0., 350:0, nil, 300:0)
; Load the other pad into the dynamic cursor buffer
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)
mypath = axlPathStart( ; Start resistor body outline
          list( -200:-100 200:-100 200:100 -200:100 -200:-100))
; Load the resistor body outline in the dynamic cursor buf
axlAddSimpleMoveDynamics(0:0 mypath "path" ?ref_point 0:0)
; Load a "path" rubberband to the first pad
axlAddSimpleRbandDynamics(8500:4500 "path"
                 ?origin 8500:4500 ?var_point -300:0)
; Load a "directline" rubberband to the second pad
axlAddSimpleRbandDynamics(9100:4500 "directline"
     ?origin 9100:4500 ?var_point 300:0)
;
mypoint = axlEnterPoint() ; Ask user for point
```

Loads two circular pads, the outline of a resistor, and rubberband connections from its pins (one with a `"path"` rubberband, the other a `"directline"` rubberband) into the dynamic cursor buffer.

The following illustration shows the cursor in a typical position as `axlEnterPoint` waits for selection of a point.

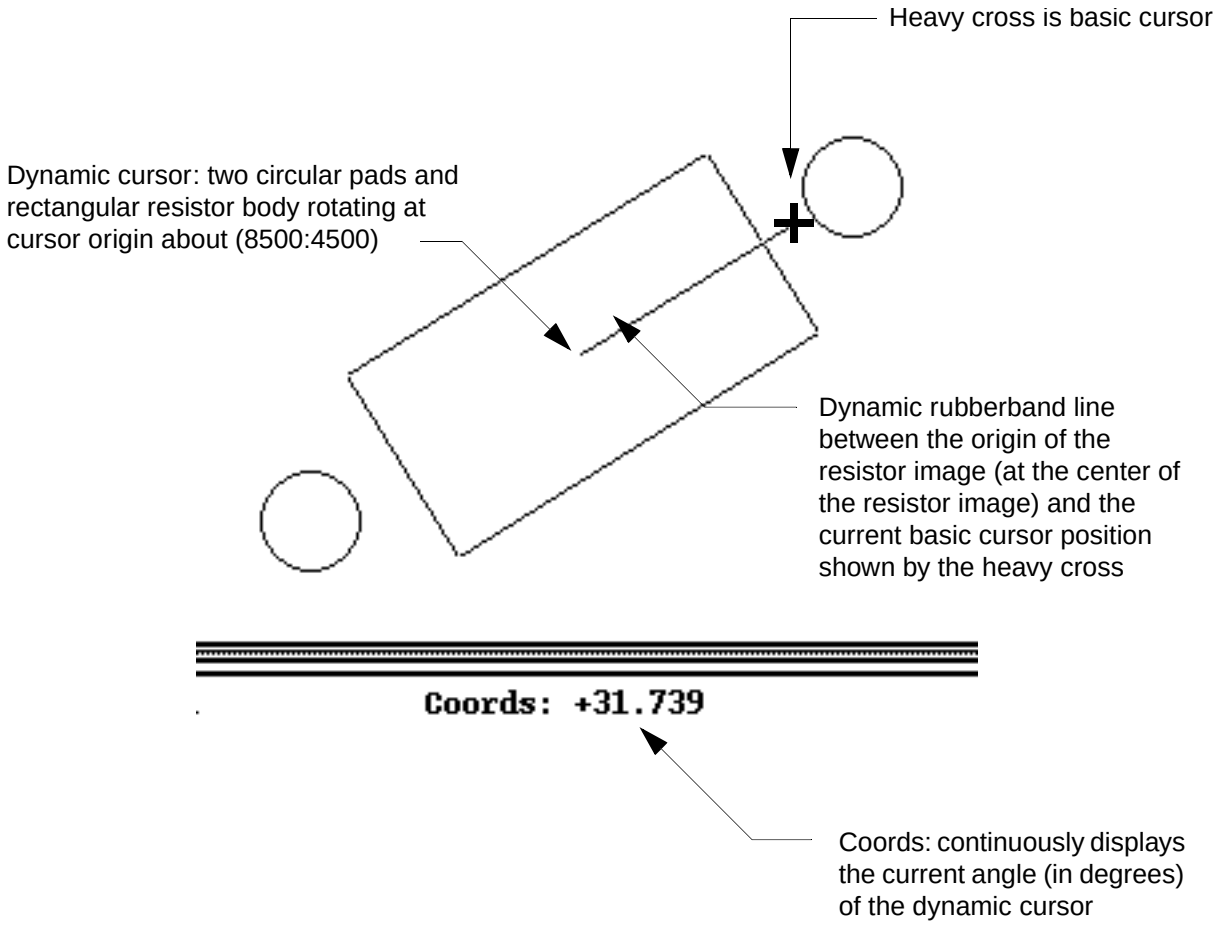


Example 2: Dynamic Cursor Rotation

```
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.

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 349.

axlAddSimpleRbandDynamics

```
axlAddSimpleRbandDynamics(
     l_fixed_point
     t_type
     ?origin        l_origin
     ?var_point     l_var_point
     ?lastPath      r_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.

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, *type*, specifies the shape of the rubberband—part of a path, direct, z-line (a combination of horizontal and vertical), arc, circle, or box.
t_type	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 *fixed_point* and *var_point*. origin and variable point of var_point `horizline`: A single horizontal line. `vertline`: A single vertical line

	`arc`: Arc betwen fixed_point and var_point. Radius varies as cursor moves circle: Circle, fixed_point is center and var_point is initial radius. "box": Add a box, fixed point is one corner and the var_point 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, fixed_point and var_point are the two endpoints.
l_origin	Cursor origin. Useful only if you plan on rotating the object, this is the center of its rotation. Also on arcs to control trangentcy. In most cases this should be nil.
l_var_point	Variable point for rubberbanding.
l_lastPath	Previous path structure. Needed to calculate tangent point if rubberbanding starts at the end of an existing path.
f_width	Optional database width of the rband. Default is 0.0.
g_color	Optional arg for defining the dynamics' color. Possible choices are:

- A layer string (i.e. class/subclass) for the layer to be used for deriving the color.
- 'ratsnestColor - the color used for ratsnest lines will be used.
- 'activeSubclassColor - the color for the active class/subclass will be used. If this changes, the color for this rband will also change.

Value Returned

`t`	Successfully added data.
`nil`	No data added.

Example

A file, demo_dynamics.il, in `<cdsroot>/share/pcb/examples/skill` demostrates the various t_type opions.

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 349.

axlAddSimpleMoveDynamics

```
axlAddSimpleMoveDynamics(
     l_origin
     r_path
     t_type
     ?ref_point     l_ref_point

)

⇒ t/nil
```

Description

Loads data incrementally into the dynamic cursor buffer. Draws path structure argument in the cursor buffer, and displays it as part of the cursor. Loading incrementally means this function adds the given data to any already in the cursor buffer.

Arguments

l_origin	Cursor origin.
r_path	Path structure containing display objects.
t_type	String specifying type of path: either `path` or `box`. Note that `path` includes lines and arcs. For this command, circles are a type of arc.
l_ref_point	Object rotation reference point.

Value Returned

`t`	Data added.
`nil`	No data added.

Example

See dynamic cursor examples in the section AXL-SKILL Interface Function Examples on page 349.

axlDesignFlip

```
axlDesignFlip(
                 g_flip
        ) t/nil
```

Description

Visually flips the design in the 'y' axis. Maintains current xy view.

Note: This command is disable if OpenGL is disabled and in certain products (e.g SIP & APD).

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

l_prompts	List containing one prompt message to display.
l_points	List of points. Returns one of these as the return value. *l_point's* only use is, if passed a point, to immediately return with the point snapped to the nearest grid.
g_gridSnap	Flag to function: `t` means snap the point according to the current grid.

Value Returned

l_point	List of coordinates, if entered. If selected, this is a list of one point.
`nil`	User did not select a point.

Example

See Example 1 in the section AXL-SKILL Interface Function Examples on page 349.

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

l_prompts	List containing one prompt message. Displays only the first string if the list contains more than one string.

Value Returned

t_string	String entered.
nil	No string entered, dialog box dismissed by clicking *Cancel*, 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

origin	Fixed point where two lines making up the angle meet.
l_prompts	List containing one prompt message.
l_refPoint	End point of a line from the *origin* that acts as the fixed line of the angle.
f_angle	Angle value in. If non-nil, does not prompt for a user-selected point.
g_lockAngle	Initial lock angle for dynamic rotation.

Value Returned

f_angle	Selected angle expressed in degrees.
nil	No angle selected.

Example

See Example 1 in the section AXL-SKILL Interface Function Examples on page 349.

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 354.

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 354.

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

point1	First pick before dynamics started.
point2	Second pick, after dynamics completes.
lastPath	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 degress 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

l_prompts	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.
l_points	List of none, one, or two points. Solicits missing points interactively using the prompts given in *l_prompts* in order.

Value Returned

l_box	List of the lower left and upper right coordinates of the box.
`nil`	Failed to get box data.

Example

```
axlDBCreateRectangle(
     axlEnterBox(?prompts
      list("First rectangle point, please..."
           "Second rectangle point, please..."))
               t "etch/top")
⇒ (dbid:12134523 nil)
```

Asks for box input to create a filled rectangle on layer `"etch/top"`.

axlEnterPath

```
axlEnterPath(
     ?prompts       l_prompts
     ?points        l_points
     ?lastPath      r_path

)

⇒ r_path/nil
```

Description

Gets the start point and subsequent points for a path, interactively with optional prompting, or from the optional argument *l_points*. Sets the start point to the first value of *l_points*, if any, and the second point to the second value, if any. If *r_path* is given, connects the dynamic rubberband to its most recent segment. Use `axlEnterPath` recursively to build up the coordinates of a path interactively.

Arguments

l_prompts	List containing one prompt message to display.
l_points	List of none, one, or two coordinates to be used as input to `axlEnterPath`.
r_path	The previously gathered part of the path. Used to calculate the tangent point for the dynamic cursor.

Value Returned

r_path	Path containing segments constructed from the combined points in *l_points* and the interactive input to `axlEnterPath`.
`nil`	Failed to get points.

Example

See the Enter Function Example on page 354.

axlHighlightObject

```
axlHighlightObject(
     [lo_dbid]
     [g_permHighlight]

)

⇒ t/nil
```

Description

Highlights the figures whose *dbids* are in *lo_dbid*.

Fewer objects support permanent highlighting than support temporary highlighting.

Note: Setting `axlDebug(t)` enables additional informational messages.

Arguments

od_dbid	List of the *dbids* of figures to be highlighted.
g_permHighlight	Distinguishes temporary highlighting from permanent highlighting using color. `t` - use PERM highlight color `nil` - use TEMP highlight color The default is `nil`.

Value Returned

`t`	Highlighted at least one figure.
`nil`	Highlighted no figures due to invalid *dbids* or objects already being highlighted.

Examples

You can use the AXL-SKILL `axlHighlightObject` and `axlDehighlightObject` functions to highlight database elements during interactive commands.

This example does the following:

a. Defines the function `highlightLoop`.

b. Loops on the function axlSelect gathering user selections to highlight.

c. Waits in a simple delay loop, then dehighlights.

You can stop the command at any time by selecting *Cancel* or *Done* from the pop-up.

```
(defun highlightLoop ()
     mypopup = axlUIPopupDefine( nil
          (list (list "Done" 'axlFinishEnterFun)
                    (list "Cancel" 'axlCancelEnterFun)))
     axlUIPopupSet( mypopup)
     axlSetFindFilter( ?enabled '("noall" "alltypes" "nameform")
                              ?onButtons "alltypes")
     (while (axlSelect)
          progn(
               axlHighlightObject( axlGetSelSet())
               ; Just a dummy delay to see what happens
                    sum = 0
                    for( i 1 10000 sum = sum + i)
                    axlDehighlightObject( axlGetSelSet())))
```

This example permanently highlights an object using color 4:

```
axlDBControl('highlightColor 4)
axlHighlightObject(axlGetSelSet() t)
```

Also see the axlHighlightObject and axlDehighlightObject Examples on page 355.

axlDehighlightObject

```
axlDehighlightObject(
     [lo_dbid]
     [g_permHighlight]

)

⇒ t/nil
```

Description

Dehighlights the figures whose *dbids* are in *lo_dbid*.

Arguments

lo_dbid	List of *dbids* of figures to be dehighlighted.
g_permHighlight	Distinguishes temporary highlighting from permanent highlighting using color. `t` - use PERM highlight color `nil` - use TEMP highlight color The default is `nil`.

Value Returned

`t`	Dehighlighted at least one figure.
`nil`	Failed to dehighlight any figures.

Example

See axlHighlightObject on page 373 for examples.

axlMiniStatusLoad

```
axlMiniStatusLoad (
     s_formHandle
     t_formFile
     g_formAction
     [g_StringOption]
     [t_restrict]

)

⇒ r_form/nil
```

Description

Loads the Ministatus form with the form file provided in this call. Replaces the current Ministatus form contents. This function is a special case of `axlForms`. See Chapter 10, "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 you ministatus form file to get that support. The "subcolor" field is optional. You should adjust the position (FLOC) of the fields to suite your form layout.

Note: Using these reserved names will also cause axlGetActiveLayer to update when user changes the layer.

```
TEXT "Active Class and Subclass:"
FLOC 1 1
ENDTEXT


FIELD class
FLOC 5 4
ENUMSET 19
```

```
OPTIONS prettyprint
POP "class"
ENDFIELD
```

option

```
FIELD subcolor
FLOC 2 7
COLOR 2 1
ENDFIELD


FIELD subclass
FLOC 5 7
ENUMSET 19
OPTIONS prettyprint ownerdrawn
POP "subclass"
ENDFIELD
```

Arguments

t_restrict — This optional argument is a string that indicates class and subclass restrictions if the form contains "class" and "subclass" popup fields that have not been overridden with calls to axlFormBuildPopup.
Possible values are:

"NONE"	- no restrictions
"TEXT"	- only layers that allow text
"SHAPES"	- only layers that allow shapes
"RECTS"	- only layers that allow rectangles
"ETCH"	- only etch layers
"ETCH_PIN_VIA"	- only etch, pin, and via layers
"ETCH_NO_WIREBOND"	- only non-wirebond etch layers

See Also

axlFormCreate on page 552 for further details.

Value Returned

r_form — Upon success, *r_form* is returned.

nil — Failure due to one of the following:

No interactive command is active or the active command is not of the type AXL registered interactive.

AXL Forms code encounters an error.

Example

See swap component example:

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

axlDrawObject

```
axlDrawObject(
     lo_dbid
)
⇒ t/nil
```

Description

Processes a list of *dbids*.

Redraws any objects that were erased by `axlEraseObject`.

Arguments

lo_dbid	List of *dbids* or one *dbid*.

Value Returned

`t`	One or more objects drawn.
`nil`	No valid *dbids* or all objects already at desired display state.

axlDynamicsObject

```
axlDynamicsObject (
     lo_dbid
     [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.

Arguments

lo_dbid	List of AXL *dbids* or single *dbid*.
l_ref_point	Optional origin point (takes cursor position if not provided).

Value Returned

t	One or more objects added to the cursor buffer.
nil	No objects added to the cursor buffer.

Example

Adds a symbol to the cursor buffer with the symbol orign as a reference point:

```
axlDynamicsObject(symbol_id, symbol_id->xy)
```

axlEraseObject

```
axlEraseObject(
     lo_dbid
)
⇒ t/nil
```

Description

Processes a list of *dbids* and erases them. Typically used with `axlDynamicsObject` to erase objects before attaching them to the cursor. Any objects erased are restored to their visibility when calling AXL shell or terminating the SKILL program.

Arguments

lo_dbid	List of *dbids* or one *dbid*.

Value Returned

`t`	One or more objects erased.
`nil`	No valid *dbids* or all objects already at desired display state.

axlControlRaise

```
axlControlRaise(
     g_option
)
⇒ t/nil
```

Description

Raises a tab in the control panel to the top. If you use this at the start of an interactive command, you override the environment variable, `control_auto_raise`.

Arguments

g_option	Supported symbols are: `'options`, `'find`, `'visibility`, and `nil`. `nil` returns a list of supported symbols.

Value Returned

`t`	Tab raised to top in control panel.
`nil`	Unknown symbol.

Example

```
axlControlRaise('options)
```

Raises the option panel to the top.

axlEnterEvent

```
axlEnterEvent(
     l_eventMask
     t_prompt
     g_snap
)
⇒ r_eventId
```

Description

A lower level event manager than other `axlEnter` functions. Provides a Skill program with more user event details. See Table 7-2 on page 384 for a list of events with descriptions.

Returns event structure containing the attributes described in Table 7-1 on page 383. Event occurrence controls what attributes are set by all event types, and sets the *`objType`* and *`time`* attributes.

Table 7-1 Event Attributes

Attribute Name	Type	Description
`objType`	string	Type of object, in this case *event*
`type`	symbol	Event occurrence
`xy`	point	Location of mouse
`xySnap`	point	Location of mouse snapped to grid.
`command`	int/symbol	Returns the callback item of `axlUIPopupDefine`
`time`	float	time stamp (seconds.milliseconds)

Note: Do not put a default handler in your case statement since the event model will change in future releases.

Table 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
`STARTDRAG`	User starts a drag operation.	
`STARTDRAG_EXTEND`	Same as `STARTDRAG` except has `extend` keyboard modifier.	
`STARTDRAG_TOGGLE`	Same as `STARTDRAG` except has `toggle` keyboard modifier.	xySnap
`STOPDRAG`	User terminated the drag operation.	
`STOPDRAG_EXTEND`	Same as `STOPDRAG` except has `extend` keyboard modifier.	
`STOPDRAG_TOGGLE`	Same as `STOPDRAG` except has `toggle` keyboard modifier.	xy, xySnap, command

Table 7-2 Events

Event	Description	Attributes/Mask
`DONE`	User requests the command to complete.	This event cannot be masked.
`CANCEL`	Respond to this event by terminating your Skill program (don't call any more *axlEnter* functions.)	This event cannot be masked.

Notes

- You will get `PICK` before `DBLPICK` events. To differentiate between a `PICK` and `DBLPICK`, highlight the selected object. Do not output informational messages or perform time consuming processing.
- Never prompt user for a double click. Instead, format prompts for the next expected event.
- Events dispatched from `axlEnterEvent` are scripted by the system if scripts are enabled.
- The *done* and *cancel* callbacks optionally defined in `axlCmdRegister` are called before the `DONE` and `CANCEL` events are returned.
- The `extend` keyboard modifier is obtained by holding the `Shift` key while performing the mouse operation.
- The `toggle` keyboard modifier is obtained by holding the `Control` key while performing the mouse operation.

Tip

You can program more easily by providing a single mask set for your command and by not attempting to change the mask set after each event.

Arguments

l_eventMask/nil	List of events to expect.
t_prompt/nil	User prompt. If nil, the default prompt is used.
g_snapGrid	If t, grid snapping is enabled while the function is active. Otherwise no grid snapping is allowed. This affects the *xySnap* value that is returned as well as dynamics and the *xy* readout. If nil, *xySnap* is not snapped to the grid and is the same as *xy*.

Value Returned

r_eventId	Event structure containing attributes.

Example

```
let( (eventMask event, loop)
     eventMask = list( 'PICK 'DBLPICK )
     loop = t
     while( loop
          event = axlEnterEvent(eventMask, nil)
          case(event->type

               ('PICK
                    ... )
               ('DBLPICK
                    ... )
               ('DONE
                    ; cleanup
```

axlEventSetStartPopup

```
axlEventSetStartPopup(
     [s_callback]

)

⇒ t/nil
```

Description

Sets a SKILL callback function called prior to a popup being displayed on the screen. Allows AXL applications to reset the popup (see `axlUIPOPUPSetsee`), thus providing context sensitive popups support.

The callback function is passed a list structure the same as the return list in `axlEnterEvent`. Use this function with `axlEnterEvent`.

The callback function is removed when an AXL application is finished. Set this at the application start, if needed.

Arguments

s_callback	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.

axlSetDynamicsMirror

sets mirror option for dynamics

```
axlSetDynamicsMirror(
            g_mirror
        ) ==> g_oldmirror
```

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

axlAddSimpleMoveDynamics

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

lo_dbid	List of *dbids* or a single *dbid*.
t_file_name	File name to use instead of a temporary file.

Value Returned

List of items describing the file created (*t_file_name x_width x_line_count*):

t_file_name	Name of the temporary file.
x_width	Width, in characters, of the widest text line.
x_line_count	Number of lines in the file.
nil	Could not create file.

axlUICmdPopupSet

```
axlUICmdPopupSet(
     r_popup
)
⇒ r_prevPopup
```

Description

Sets up a popup menu with all menu items required throughout the execution of the command. Call during the command's initialization process. Use of this procedure modifies the behavior of `axlUIPopupSet` so that it makes unavailable all popup items not in the defined popup.

Adds a `cmdPopupId` property to AXL user data which restores popup entries whenever the AXL command state is restored. The command popup is cleared when the Skill command ends.

Arguments

r_popup	Popup handle, obtained by calling `axlUIPopupDefine`. A `nil` value turns off this popup.

Value Returned

r_prevPopup	Popup set previously defined.

Note: This procedure does the same as `axlCmdPopupSet` for non-WXL UI's.

axlZoomToDbid

```
axlZoomToDbid(
     o_dbid/lo_dbid
     g_always

)

⇒ t/nil
```

Description

Processes a list of *dbids* and centers and zooms the display around them. Zoom is done so objects extents fill about 20% of the display.

Note: You should highlight the objects.

Note: If more than 20 objects are passed no zoom is done.

Arguments

o_dbid	List of *dbids* or one *dbid*.
g_always	If `t`, then ignores `NO_ZOOM_TO_OBJECT` environment variable.

Value Returned

`t`	One or more objects zoomed.
`nil`	No valid *dbids* or all objects are already at desired display state.

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.

Note: Caution: Passing an illegal format may result in a bad return.

Arguments

*(l_seg1 l_seg2 ...) g_clockwise*Each *l_seg* is:

(l_startPoint l_endPoint [f_width] [l_center] [f_radius])

where

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 ⇒ 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)))
```

8

Allegro PCB Editor Command Shell Functions

Overview

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/lt_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 funckey settings.

Arguments

t_alias	Name of the Allegro PCB Editor environment alias.

Value Returned

t_value	String value of the Allegro PCB Editor environment alias.
lt_names	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

t_alias	Name of the environment funckey.
nil	Returns list of all current funckeys

Value Returned

t_value/nil	String value of the environment funckey. Returns *nil* if the funckey is not set.
lt_names	If passed *nil,* returns list of funckeys names.

See Also

axlSetFunckey, axlGetAlias

EXAMPLE 1

Gets the value of the funckey assigned to shifted function key m.

```
alias = axlGetFunckey("m")
==> "grid"
```

EXAMPLE 2

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.

Important

Variable names and values can change from release to release.

Arguments

t_variable	String giving name of the Allegro PCB Editor environment variable.

Value Returned

t_value	List containing string value of the Allegro PCB Editor environment variable.
`nil`	Variable not set.

Example 1

```
menu = axlGetVariable("menuload")
                    ==> "geometry"
            psmpath = axlGetVariable("psmpath")
                    ==> ". symbols"
```

- Gets value of the current menu loaded.

  Variable name is *menuload*.

- 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 Skill's parseString to break them back to the component pieces will not give the correct result.

Note: Variable names are case insensitive.

Note: Variable names and values can change from release to release.

Arguments

t_variable	Name of the Allegro PCB Editor environment variable.
nil	If nil, returns a list of all set variables in Allegro PCB Editor.

Value Returned

t_value	String value of the Allegro PCB Editor environment variable.
nil	Returns nil if the variable is not set.
lt_names	If passed, nil returns list of variable names

See Also

axlGetVariable

Example

Gets the value of the Package Symbol search path:

```
path = axlGetVariable("psmpath")
==> ( "." "symbols" "/cds/root/share/pcb/allegrolib/symbols")
```

axlJournal

```
axlTempFile(
          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

Typically the journal file is buffered. Reading the file while it is open may be unpredictable.

Argument

g_mode	see above

Value Returned

See above

Example

Name of file

```
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

t_alias	Name of the Allegro PCB Editor environment alias.
t/nil	t protects the alias, and nil unprotects the alias.

Value Returned

t	Successfully protected or unprotected the alias.
nil	Alias is not set, or the function received invalid data.

Example

```
axlProtectAlias( "F2" t)
```

Protects the F2 function key.

axlIsProtectAlias

```
axlIsProtectAlias(
     t_alias
)
⇒ t/nil
```

Description

Tests if the alias is read-only (or writeable). 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 a Allegro PCB Editor environment variable. When you set a variable as read-only, it cannot be changed.

Note: Variable names are case insensitive.

Arguments

t_variable	The name of the Allegro PCB Editor environment variable.
g_Enable	`t` to set read-only; `nil` to make writable and do not provide if using to test variable read-only state.

Value Returned

`t/nil`	In query mode (*no g_Enable* option) returns `t` if variable is read-only and `nil` if not. If changing the read-only mode, returns `t` if successful and `nil` if variable is not currently set.

See Also

`axlGetVariable`

Examples

The following example:

- Sets `psmpath` to read-only.
- Queries the setting.
- Resets `psmpath` to writable.

Query the setting:

```
     axlReadOnlyVariable("psmpath" t)
     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 `axlSetFunckey, 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

```
axlSetAlias( "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 `axlSetFunckey, 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

```
axlSetAlias( "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

t_alias	name of the Allegro environment alias.
g_value	Value to which the environment alias is to be set. Can be a string, or nil.

Value Returned

t	Returns *t* if successful.
nil	Returns *nil* 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

```
axlSetFunckey( "m" nil)
```

axlSetVariable

```
axlSetVariable(
     t_variable/lt_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*. *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 (*lt_variable*).

Variable names and values can change from release to release.

Arguments

t_variable	String giving the name of the Allegro PCB Editor environment variable.
g_value	Value to which the environment variable is to be set. Can be a string, int, `t`, or `nil`.

Value Returned

`t`	Environment variable set.
`nil`	Environment variable not set.

Example

Sets new library search path `libpath`.

```
(axlSetVariable "libpath" "/mytools/library")
⇒ t
libraryPath = (axlGetVariable "libpath")
⇒ "/mytools/library"
```

Using list mode.

```
axlSetVariable("psmpath" '("." "symbols"))
                  ==> t
             axlGetVariableList("psmpath")
                  ==> ("." "symbols")
```

axlShell

```
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.

Important

This function might not be portable across Allegro PCB Editor releases.

Arguments

t_command	Allegro PCB Editor shell command or commands.

Value Returned

t	Always returns t.

See Also

axlShellPost

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:

axlShellPost

```
axlShellPost(
t_command
) ==> t
```

Description

This works similar to `axlShell` 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 the 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

t	Always returns *t*.

See Also

axlShell

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`.

Example

```
(axlUnsetVariable "libpath")
⇒ "/mytools/library"
libraryPath = (axlGetVariable "libpath")
⇒ nil
```

Clears the library path `libpath` when its current value is `/mytools/library`.

9

User Interface Functions

Overview

This chapter describes the AXL/SKILL functions you use to confirm intent for an action, prompt for text input, display ASCII text files, and flush pending changes in the display buffer.

Window Placement

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 `inner`.
`Canvas` or `Window`	Uses the canvas (drawing) area or the entire window for the placement rectangle.The default is `Window`.
`Border` or `NoBorder` (Default `Border`)	Leaves a slight border around the placed window. If `noborder` is set, the window is set directly against the placement rectangle. The default is `Border`.
`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 differnt tools such as APD, SIP 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
```

Caution

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.

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 9-1 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`	Parition menu for PCB products
`specctraquest.men`	PCB SI menu
`apd.men`	APD menu
`sip.men`	SiP menu
`icp_symbol.men`	APD/SIP symbol editor menu
`apd_partition.men`	APD Parition editor menu
`sip_partition.men`	SIP Parition 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/SIP Free Viewer
`viewlayout.men`	Allegro Viewer Plus

Menu Terms

- Menubar - 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 menbar) 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 menubar.
- All menubar items should be submenus. Do not add a command menu item at this level.
- Do not add excessive items to the menubar. If the menubar displays on two lines for a typical window width you may have too many items.
- Keep the display text relatively short, especially on the menubar.

MENU CUSTOMIZATION METHODS 

- Provide own customization menu via CDS_SITE. Replace the Cadence provided menu (.men file) with your own.
  - Advantages: Relatively easy and no Skill programming required.
  - Disadvanges: 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 is immune to many Cadence menu changes.
  - Disadvanges: 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 flexibity as overriding the default menu file including targeting specific menus.
  - Disadvanges: Need to examine your Skill code with 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_pcb`* and *allegro_viewer*.

Tools support dynamically (via Skill) modifing 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:

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 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.


>              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:
```

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.

Example 3 - Simple Menu Example

```
          DISPLAY (indents reflect the various pulldown levels)
               File Help
                    Open Contents
                    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

See `axlCancelOn`.

axlCancelOn

```
axlCancelOn(
)
⇒ t
axlCancelOff(
)
⇒ t
axlCancelTest(
)
⇒ t/nil
```

Description

Allows Skill code to test for when a user clicks *Cancel*.

When cancel is enabled, the traffic light is yellow.

Although you can nest cancel calls, you should make an equal number of cancel off calls as cancel on calls.

Note: To avoid problems, always place the cancel on/off call pairs in the same function.

These calls do not work from the Skill or 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

Only `axlCancelTest` returns meaningful data.

`t`	User click cancel.
`nil`	User did not click cancel.

Examples

```
count = 0
axlCancelOn()
while ( count < 50000 && !axlCancelTest()
     printf("Count = %d\n" count)
     count++
)
axlCancelOff()
```

axlCancelTest

See `axlCancelOn`.

axlMeterCreate

```
axlMeterCreate(
          t_title
          t_infoString
          g_enableCancel
          [t_formname]
          [t_infoString2]
          [g_formCallback]
-> t/nil
```

Description

Starts progress meter with optional cancel featur

Note: Always call `axlMeterDestroy` when done with meter.

Arguments

t_title	Title bar of meter.
t_infoString	One line of 28 characters used for anything you want (can be updated at meter update).
g_enableCancel	`t` enable the application *Stop* button on graphical UI-based applications. When enabled and the user picks the *Stop* button, a true is returned by the call to `axlMeterIsCancelled()`.
t_formname	(Optional) The name of an alternate form that can be used with these functions which has an info field named *progressText* and a progress field named *bar*. axlMeterIsCancelled will also notice if a *Cancel* menu button has been pressed. If you do not give a form name `axlprogress.form` will be used.
t_infoString2	(Optional) By Default "".
g_formCallback	(Optional) The name of a Callback function that you want called for any buttons or fillins etc you may have on your form. This works the same as `g_formAction` in `axlFormCreate`.

Value Returned

`t`	On success; otherwise `nil`.

See Also

axlMeterCreate, axlMeterIsCancelled, axlMeterDestroy and `axlFormCreate`

Example

```
axlMeterCreate("SigNoise Design Audit", "", t)
total = <total nets>
done = 0
while(<still next net> && (!axlMeterIsCancelled())
          < do work >
          axlMeterUpdate( (100 * ++done)/total
               sprintf(nil "Check %d of %d nets" done total))
)
axlMeterDestroy()
```

axlMeterDestroy

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

Description

Closes the progress meter form and shuts off Cancel mode if enabled.

Arguments

None

Value Returned

`t`	If meter was destroyed; otherwise `nil`.

See Also

`axlMeterCreate`

axlMeterUpdate

```
axlMeterUpdate(
          x_percentDone
          t_infoString
          [t_infoStr2]
) -> t/nil
```

Description

Updates progress meter bar and/or info text. The percent done and/or the info string may be updated.

Arguments

x_percentDone	Integer task percent done (0-100)
t_infoString	Update text for progress meter info text line. Value is one of: nil - leave info text as it is. "" - clear info string field.
newText	Update field with new text.
t_infoStr2	(optional) Text for second line.

Value Returned

t	On success; otherwise nil.

See Also

axlMeterCreate

axlMeterIsCancelled

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

Description

If cancel was enabled at meter creation, the status of cancel is returned (`t` if cancelled; otherwise `nil`).

If a field named *Cancel* was hit, it is cancelled

Arguments

None

Value Returned

`t`	If meter was cancelled; otherwise `nil`.

See Also

`axlMeterCreate`

axlUIMenuLoad

```
axlUIMenuLoad (
     t_menufile

)⇒ t_previousMenuName/nil
```

Description

Loads the main window menu from the file *t_menuFile*. Adds a default menu file name extension if *t_menuFile* has none. The MENUPATH environment variable is used to locate the file if *t_menuFile* does not include the entire path from the root drive.

Note: The intent of this procedure is to allow a custom menu to be loaded for debugging purposes.

Arguments

t_menuFile	Name of the file to which the menu is dumped. If *t_menuFile* is nil, 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

t_previousMenuName	Name of the previous menu.
nil	Menu not be located.

See Also

axlUIMenuFind

axlUIMenuDump

```
axlUIMenuDump (
     t_MenuFile

) ⇒ t_previousMenuName/nil
```

Description

Dumps the main window's current menu to the file *t_menuFile*. Adds default menu file name extension if *t_menuFile* has none.

Notes:

- There is no user interaction when an existing file is overwritten.
- This function is for the Windows-based GUI only.

Arguments

t_menuFile	Name of the file to which the menu is dumped. If *t_menuFile* is `nil`, 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

t_previousMenuName	Full name of the file that is written.
`nil`	No file is written.

axlUIColorDialog

```
axlUIColorDialog(
                  r_window/nil
                  l_rgb
         ) -> l_rgb/nil
```

Description

Invokes standard color selection dialog box. You must provide a parent window, 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

r_window	Parent window. If `nil`, use main program window. Return handle of `axlFormCreate` is of type *r_window*.
l_rgb	Seeded red, green, or blue.

Value Returned

l_rgb	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(
     axlUIConfirm(
                  t_message
                  [s_level]
                  )
          ==> t
```

Description

Displays the string *t_message* in a dialog box.

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

t_message	Message string.
s_level	Option level symbol; default is info level, other levels are `warn` and `error`.

Value Returned

`t`	Always returns `t`.

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

axlUIControl

```
axlUIControl(
                  s_name
                  [g_value]
          )
          ==> g_currentValue/ls_names
```

Description

Inquires and sets the value of graphics. If setting a value, the return is the old value of the control. If an active form displaying the current settings is open, it may not update. Settings currently supported:

```
{UIScreen, NULL, "screen" },
{UIVScreen, NULL, "vscreen"},
{UIMonitors, NULL, "monitors"},
{UIPixel2DB, NULL, "pixel2UserUnits"},
Name:   screen
Value:  (x_width x_height)
Set?:   No

Description: Retrieves the screen's width and height in pixels
Equiv:  none
Side Effects: none

Name:   vscreen
Value:  (x_width x_height)
Set?:   No
Description: Retrieves the screen's virtual width and height in pixels. This will not be the same as
'screen if running Windows XP and enabled monitor spanning option. Also requires multiple monitors and
graphic card(s) capable of supporting multiple monitors.
Equiv:  none
Side Effects: On UNIX always returns the same size as screen.

Name:   monitors
Value:  x_number
Set?:   No
Description: Retrieves the number of monitors available.
Equiv:  none
Side Effects: On UNIX always returns 1 since we currently do not support multi-monitors on UNIX.

Name:   pixel2UserUnits
Value:  f_number
Set?:   No
Description: Returns number user units per pixel taking into account the current canvas size and zoom
factor. Changes with the current zoom factor.
Equiv:  none
Side Effects: none
```

Arguments

s_name	Symbol name of control. `nil` returns all possible names.
s_value	Optional symbol value to set. Usually a `t` or a `nil`.

Value Returned

ls_names	If name is `nil` then returns a list of all controls.

Examples:

Get screen size:

```
size = axlUIControl('screen)
-> (1280 1024)
```

Get pixel to user units:

```
axlUIControl('pixel2UserUnits)
-> 17.2
```

axlUIMenuChange

```
axlUIMenuChange(
     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 assi command text to a submenu.

Note: See discussion in axlUIMenuFind about menu changes.

Arguments

`x_menuId` The menuId from axlUIMenuFind

`s_option/g_mode pairs` See Table 10-2 on page 518

Value Returned

`t`, if menu item is changed, and `nil` if the command failed to change the menu item.

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

axlUIMenuDelete

```
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

axlUIMenuFind

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 menubar)

```
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 menubar). 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 menubar 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 menubar (`menuId = nil`) or sub-menu.

   The `g_menuOption` when used in location mode returns the top or bottom of the indexed sub-menu (see below).


Tip

Examples shown below provide typical uses.


Caution

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.
- Menus may be reorganized so expecting to find a command on a particular sub-menu 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 recommandations.

Arguments

`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 by just a command but is realy a command line. For example, if the command is register 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 of bottom of the menu where the command exists. Bottom option also indicates to axlUIMenuInsert 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 succesful 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 menubar):

```
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_subMenuId/nil
```

- Command to add Sub-menu end (optional)

```
axlUIMenuInsert(
             x_menuId
             'end
         ) -> t/nil
```

- Command to add multipleitems

```
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 with dispatchs 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 axlUIMenuFind 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: `separator` - add a separator (horizontal line) `popup` - create a new submenu
`t_command`	command to run ■ this is ignored for a 'separator ■ this is the display string for 'popup option 'end ■ pops the menu stack if creating a menu tree

`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 separtor after 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 2 cmmmands

```
q = axlUIMenuFind( nil -1)
z = axlUIMenuInsert(nil 'popup "MyMenu")
z = axlUIMenuInsert(nil "1" "echo hello 1" )
z = axlUIMenuInsert(nil "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.

- See axlUIMenuFind for cautions about portability across releases.
- When multiple menu registers are done, there may be depedancies. 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.

Arguments

`t_command`	Command to insert menu before (see axlUIMenuFind)
`x_location`	Location before to insert menu (see axlUIMenuFind)
`ll_menu`	List of menu items to load (see format 3 option of axlUIMenuInsert)
`g_menuOption`	Indication to add to top or bottom of menu (see axlUIMenuFind)

Value Returned

`t`, if register function for indicated callback, `nil`, if the command failed to register trigger

See Also

axlUIMenuFind, axlUIMenuInsert, axlTriggerSet, axlUIMenuDebug

Example

See `<cdsroot>/share/pcb/examples/skill/ui/menu.il`

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

t_message	Message string displayed.
t_default	Default value displayed to the user and returned if user presses only the *Return* key or clicks *OK*.
'password:	Obscure and do not script user input.

Value Returned

t_response	User response or default value.
nil	User selected *Cancel*.

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 (e.g. brd). Via Skil,l 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 (e.g. File Browser dialogs) cannot be closed via this call.

Arguments

None

Value Returned

t	always

See Also

axlUIWPerm

axlIsViewFileType

```
axlIsViewFileType(
     g_userType
)
⇒ t/nil
```

Description

Tests whether *g_userType* is a long message window type.

Arguments

g_userType	Argument to test.

Value Returned

t	*g_userType* is of type *r_windowMsg*.
nil	*g_userType* is not of type *r_windowMsg*.

Example

```
logWindow =
     axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)
     axlIsViewFileType(logWindow)
⇒ t
```

- Creates a window using axlUIViewFileCreate (See axlUIViewFileCreate on page 465.)
- Tests whether the window is a view file type.
- Returns t.

axlUIViewFileCreate

```
axlUIViewFileCreate(
     t_file
     t_title
     g_deleteFile
     [lx_size]
     [lt_placement]
)
⇒ r_windowMsg/nil
```

Description

Opens a file view window and displays the ASCII file *t_file* using the arguments described below.

Arguments

t_file	Name of the ASCII file to display.
t_title	Title to be display in window title bar.
g_deleteFile	Deletes the file when the user quits the window or another task reuses the window. Use this to delete files you want to discard, since this allows correct operation of the *Save* and *Print* buttons in the file view window.
lx_size	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.
lt_placement	Location of the window with respect to the Allegro PCB Editor window. The default location is centered on Allegro PCB Editor window.

Value Returned

r_windowMsg	Window *r_windowMsg*.
nil	*r_windowMsg* not displayed.

Example

```
logWindow =
     axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)
⇒ windowMsg:2345643
```

- Displays the batch DRC log file, saving the window id.
- Deletes the file `drc.log` when the user exits the window.
- The log file displays in a window.
- When the user chooses *Close*, deletes the file `batch_drc.log`.

axlUIViewFileReuse

```
axlUIViewFileReuse(
     r_windowMsg
     t_file
     t_title
     g_deleteFile
)
⇒ t/nil
```

Description

Reuses the existing view window *r_windowMsg* created by an earlier call to the function axlUIViewFileCreate, gives it the title *t_title*, and displays the ASCII file *t_file* in the window. If the user has already quit the window since its previous use, the window reopens at its old size and position.

Arguments

r_windowMsg	*dbid* of the existing view window created earlier with axlUIViewFileCreate.
t_file	Name of the ASCII file to display.
t_title	Title to display in window title bar.
g_deleteFile	Deletes file when the user quits the window or another task reuses the window. Use to delete files you want to discard, since this allows correct operation of the *Save* and *Print* buttons in the file view window.

Value Returned

t	File displayed.
nil	File not displayed.

Example

```
(axlUIViewFileReuse logWindow "ncdrill.log" "NC Drill Log" t)
```

- Displays the file `ncdrill.log`, reusing the window `logWindow` created when displaying `router.log` in the `axlUIViewFileCreate` example.
- Exiting the window deletes the file `ncdrill.log`.

axlUIYesNo

```
axlUIYesNo(
                t_message
                [t_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

t	User responded *Yes*.
nil	User responded *No*.

See Also

axlUIConfirm

Examples

The following examples are a typical overwrite question.

```
            axlUIYesNo( "Overwrite module?" )

            axlUIYesNo( "Overwrite module?" nil 'no )

            axlUIYesNo( "Overwrite module?" "My Skill Program" )

        A confirmer window is displayed. If the user selects Yes, the
        function returns t, otherwise it returns nil.
**/

list
axlUIYesNo(int argc, list *argv)
{
    char *str, *title;
    int dflt;

    str = axluGetString(NULL, argv[0]);
    title = (argc>1) ? axluGetString(NULL, argv[1]) : NULL;
    dflt = (argc>2) ? DfltResponse(argv[2]) : MN_YES;

    return(MNYesNoWTitle(str, title, dflt) ? ilcT : ilcNil);
}

 /*
#ifdef DOC_C
```

axlUIWExpose

```
axlUIWExpose(
     r_window/nil
)
⇒ t/nil
```

Description

Opens and redisplays a hidden or iconified window, bringing it to the front of all other current windows on the display. If `nil`, the main window is displayed.

Arguments

r_window	Window *dbid*.

Value Returned

`t`	Window opened and brought to front.
`nil`	*dbid* was not of a window.

Example

```
logWindow =
     axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)
; Other interactive code, possibly
; causing Batch DRC Log window to be covered
; Uncover the log window:
axlUIWExpose(logWindow)
⇒t
```

- Displays a window using `axlUIViewFileCreate`.
- Interactively moves window behind one or more other windows using the *back* selection of your window manager.
- Calls `axlUIWExpose`.

  Window comes to the top above all other windows.

axlUIWClose

```
axlUIWClose(
     r_window
)
⇒ t/nil
```

Description

Closes window *r_window*, if it is open.

Arguments

r_window	Window *dbid*.

Value Returned

t	Window closed.
nil	Window already closed, or *dbid* is not of a window.

Example

```
logWindow =
     axlUIViewFileCreate("batch_drc.log" "Batch DRC Log" t)
; Other interactive code
;
axlUIWClose(logWindow)
⇒t
```

- Displays window using `axlUIViewFileCreate`.
- Closes window using `axlUIWClose`.

axlUIWPrint

```
axlUIWPrint(
     r_window/nil
     t_formatString
     [g_arg1 ...]
)
⇒ t/nil
```

Description

Prints a message to a window other than the main window. If *r_window* does not have a message line, the message goes to the main window. This function does not buffer messages, but displays them immediately. If the message string does not start with a message class (for example \e), it is treated as a text (\t) message. (See axlMsgPut on page 668) If `nil`, displays the main window.

Arguments

r_window	Window *dbid*.
t_formatString	Context message (`printf`–like) format string.
g_arg1...	Any number of substitution arguments to be printed using *t_formatString*. Use as you would a C–language `printf` statement.

Value Returned

`t`	Message printed to window.
`nil`	*dbid* is not of a window.

Example

```
axlUIWPrint(nil "Please enter a value:")
Please enter a value:
⇒t
```

Prints a message in the main window.

axlUIWRedraw

```
axlUIWRedraw(
     r_window/nil
)
⇒ t/nil
```

Description

Redraws the indicated window. If the window *dbid* is `nil`, redraws the main window.

Arguments

r_window	Window *dbid* or, if `nil`, the main window.

Value Returned

`t`	Window is redrawn.
`nil`	*dbid* is not of a window.

axlUIWBlock

```
axlUIWBlock(
     r_window
)
⇒ t/nil
```

Description

This function is not compatible with the g_nonBlock = nil option to axlFormCreate. If using this function with axlFormCreate you must set a callback on the g_formAction.

This places a block on the indicated window until it is destroyed. All other windows are disabled. It may be called recursively, unlike the block option in `axlFormCreate`.

Once you enter a blocking mode you should not bring up a window that is non-blocking. This behavior is not defined and is not supported.

If you block, you should set the block attribute `block` in the Window Placement list *`lt_placement`* so that the title bar shows it is a blocking window.

If you have a window callback registered you must allow the window to close since the unblock facility unblocks other windows upon close so that the correct window will get the focus after the blocked window is destroyed.

Note: You should set the block symbol option using the *lt_placement* option in the function that creates the window to visually indicate that the window is in blocking mode.

Arguments

`r_window`	Window *`dbid`*.

Value Returned

`t`	Success
`nil`	Failure (For example, the window is closed or the *`dbid`* is not of a window).

axlUIEditFile

```
axlUIEditFile(
     t_filename
     t_title/nil
     g_block
)
⇒ r_window/t/nil
```

Description

Allows the user to edit a file in an OS independent manner (works under both UNIX and Windows.)

User may override the default editor by setting either the `VISUAL` or `EDITOR` environment variables.

Windows notes

- The default editor is *Notepad*.
- The title bar setting is not supported.

Unix notes

- The default editor is `vi`.
- An additional environment variable, `WINDOW_EDITOR`, allows the user to specify an X-based editor such as `xedit`. The title bar is not supported in this mode.

Note: In blocking mode, the windows of the main program do not repaint until the file editor window exits.

Only `axlUIWClose` supports the *`r_window`* handle returned by this function.

Arguments

`t_filename`	Name of file to edit.
`t_title`	Title bar name, or `nil` for default title bar.
`g_block`	Flag specifying blocking mode (`t`) or non-blocking mode (`nil`).

Value Returned in Non-blocking Mode

r_window	Success
nil	Failure

Value Returned in Blocking Mode

t	Success
nil	Failure

axlUIMultipleChoice

```
axlUIMultipleChoice(
     t_question
     lt_answers
     [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

t_question	Text of the question for display.
lt_answers	A list of text strings that represent the possible answers.
t_title	Optional title. If not present, a generic title is provided.

Value Returned

x_answer	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.
nil	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.

Arguments

r_windowMsg	Existing view window.
x_line	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

x_lines	Number of lines in the view window.
`nil`	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 it 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

r_window	Window id.
t/*nil*	`t` - set permanent
	`nil` - reset permanent.

Value Returned

`t`	Window exists.
`nil`	Window does not exist.

Example 1

```
handle = axlFormCreate('testForm "axlform" nil 'testFormCb, t nil)
axlUIWPerm(handle t)
```

Opens a test form and makes it permanent.

Example 2

```
ret = axlUIWPerm(handle)
```

Tests whether the window is permanent.

axlUIWSetHelpTag

```
axlUIWSetHelpTag(
     r_window
     t_tag

)

⇒ t/nil
```

Description

Attaches the given help tag to a pre-existing dialog with a port. This function supports subclassing of the help tags, that is, if a help tag is already associated with the dialog, it will not be replaced. This functions adds the new help tag. Adding a new help tag to a pre-existing one is done by concatenating the two with a dot.

For example:

Pre-existing Help Tag:	myOldTag
New Help Tag:	myNewTag
Resulting Help Tag:	myOldTag.myNewTag

Arguments

r_window	Window id.
t_tag	Subclass of the help string.

Value Returned

t	Help tag attached.
nil	Invalid arguments.

axlUIWSetParent

```
axlUIWSetParent(
     o_childWindow
     o_parentWindow/nil

)

⇒ t/nil
```

Description

Sets the parent of a window. When a window is created, its parent is the main window of the application, which is sufficient for most implementations. To run blocking mode on a form launched from another form, set the child form's parent window to be the launched form.

Setting the parent provides these benefits:

- Allows blocking mode to behave correctly.
- If the parent is closed, then the child is also closed.
- If the parent is iconified, then the child is hidden.
- The child stays on top of its parent in the window stacking order.

Arguments

o_childWindow	Child window handle.
o_parentWindow	Parent window (if `nil`, then the main window of the application which is normally the default parent.)

Note: A parent and child cannot be the same window.

Value Returned

`t`	Parent is successfully set.
`nil`	Could not set the parent due to an illegal window handle.

axlUIWShow

```
axlUIWShow(
     r_window/nil
     s_option
)
⇒ t/nil
```

Description

Shows or hides a window depending on the option passed. If the window id passed is `nil`, the function applies to the main window.

Notes:

- Using the `showna` option on a window may make the window active.
- Using the `show` option on a window that is already visible may not make it active.

Arguments

r_window	The window id. If `nil`, signifies the main window.
s_option	One of the following:

`'show`	Show and activate the window
`'showna`	Show but don’t activate the window.
`'hide`	Hide the window.
`nil`	Show available options.

Value Returned

`t`	Window shown or hidden.
`nil`	Window id not correct or an invalid option given.

axlUIWTimerAdd

```
axlUIWTimerAdd(
     o_window
     x_timeout
     g_oneshot
     u_callback
)
⇒ o_timerId/nil
```

Description

Adds or removes a callback for an interval timer.

This is not a real-time timer. It is synchronous with the processing of window based messages. The actual callback interval may vary. The timer does not go off (and call you back) unless window events for the timer window (*o_window*) are being processed. You must be waiting in a UI related call (for example, `axlEnter*`, a blocking `axlFormDisplay`, `axlUIWBlock`, etc.)

To receive callbacks return to the main program message processing. Another window in blocking mode, however, can delay your return to the main program.

You may add properties to the returned *timerId* to store your own data for access in your timer callback as shown:

```
procedure( YourSkillProcedure()
     ; set up a continuous timer using the main window
     timerId = axlUIWTimerAdd(nil 2000 nil 'YourTimerCallback)

timerId->yourData = yourdata
)
procedure( YourTimerCallback( window timerId elapsedTime)
     ;your time period has elapsed. do something.
)
```

Arguments

o_window	The window the timer is associated with. If *o_window* is `nil`, the timer is associated with the main window.
x_timeout	Timeout in milliseconds before the timer is triggered and calls your callback procedure. Timeout is not precise because it depends on processing window messages.

g_oneshot Controls how many times the timer triggers. Use one of these values:

`t` - Timer goes off once and automatically removes itself.
`nil` - Timer goes off at the set time interval continuously until it is removed by `axlUIWTimerRemove`.

u_callback Procedure called when the timer goes off. Called with these arguments with its return value ignored:

```
u_callback(
    o_window
    o_timerId
    n_elapsedTime
)
```

o_window Window you provided to `axlUIWTimerAdd`

o_timerId Timer id which returned by `axlUIWTimerAdd`.

x_elapsedTime Approximate elapsed time in milliseconds since the timer was added.

Value Returned

o_timerId The identifier for the timer. Use this to remove the timer. This return value is subject to garbage collection when it goes out of scope. When the garbage is collected, the timer is removed. Don't count on garbage collection to remove the timer, however, because you do not know when garbage collection will start. If you need a timer that lasts forever, assign this to a global variable.

`nil` No timer added.

axlUIWTimerRemove

```
axlUIWTimerRemoveSet(
     o_timerId
)
⇒ t/nil
```

Description

Removes a timer added by `axlUIWTimerAdd`.

Arguments

o_timerId	Id returned by `axlUIWTimerAdd`.

Value Returned

`t`	Timer removed.
`nil`	Timer id invalid.

axlUIWUpdate

```
axlUIWUpdate(
     r_window/nil
)
⇒ t/nil
```

Description

Forces an update of a window. If you made several changes to a window and are not planning on going back to the main loop or doing a SKILL call that requires user interaction, use this call to update a window. You could use this, for example, if you are doing time-consuming processing without returning control to the UI message pump.

Note: This is required for Bristol based code. In other implementations it has no effect.

Arguments

r_window	Window id or `nil` if the main window.

Value Returned

`t`	Window updated.
`nil`	Window already closed or invalid window id.

axlUIYesNoCancel

```
axlUIYesNoCancel(
     t_message
     [t_title]
     [s_default]
)
⇒ x_result
```

Description

Displays a blocking *Yes*/*No*/*Cancel* dialog box with the prompt message provided.

Arguments

t_message	Message to display.
t_title	Optional. What to put in the title bar of confirm. The default is the program display name.
s_default	Optional. May be either `yes`, `no` or `cancel` to specify default response. The default is `yes`.

Value Returned

x_result	Number based on the user's choice: `0` for *Yes* `1` for *No* `2` for *Cancel*

Examples

axlUIDataBrowse

```
axlUIDataBrowse(
     s_dataType
     ls_options
     t_title
     g_sorted
     [t_helpTag]
     [l_callback]
     [g_args]
)
⇒ lg_return
```

Description

Analyzes all objects requested by the caller function, passing each through the caller's callback function. Then puts the objects in a single-selection list.

This list blocks until a user makes a selection. Once the user selects an object, it is passed back to the caller in a list containing two objects: the selected name and, for a database object, the AXL dbid of the object.

Arguments

s_dataType 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
```

ls_options 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 checkbox for the database
`'LIBRARY_FIXED`	Read-only checkbox for files (library)

t_title	Prompt for the title of the dialog
g_sorted	Switch indicating whether or not the list should be sorted
t_helpTag	Help tag for the browser
l_callback	Callback filter function which takes the arguments name, object, and g_arg passed in. Returns t or nil based on whether or not the object is eligible for browsing.
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)
```

10

Form Interface Functions

Overview

This chapter describes the field types and functions you use to create Allegro PCB Editor forms (dialogs).

Allegro PCB Editor AXL forms support a variety of field types. See Callback Procedure: formCallback on page 581 and Using Forms Specification Language on page 505 for a complete description of field types.

See Also

axlFormCreate - open a form

axlFormCallback - callback model for interaction with user

axlFormBNFDoc - Backus Naur Form, form file syntax

demos

axlFormTest

Programming

It is best to look at the two form demo.

- basic controls -- `axlform.il/axlform.form`
- grid control - `fgrid.il/fgrid.form`

The first step is to create form file. Use `axlFormTest` to insure fields are correctly postioned.

The following procedure is generally used.

1. Open form (`axlFormCreate`)

2. Initialize fields (`axlFormSetField`)
3. Display Form (`axlFormDisplay`)
4. Interactive with user (`axlFormCallback`)
5. Close Form (`axlFormClose`)

 *Tip*

- Many users find that it is easier to distribute their program using a form if they embed the form file in their Skill code. In this case use Skill to open a temporary file and print the statements, open for form, then delete the file.
- Use `axlFormTest`("*<form file>*") to interactively adjust of fields.
- You can use "`ifdef`", "`ifndef`", and Allegro environment variables (`axlSetVariable`) to control appearance of items in the form file.

Field / Control

Most interaction to the controls are via axlFormSetField, axlFormGetField, axlFormSetFieldEditable, and axlFormSetFieldVisible.Certain controls have additional APIs which are noted in the description for the control.

Most controls support setting their background and foreground colors. See `axlColorDoc` and `axlFormColorize` for more information.

Following is a list of fields and their capabilities.

TABSET / TAB

A property sheet control. Provides the ability to organize and nest many controls on multiple tabs.

Unlike other form controls you nest other form controls within TAB/ENDTAB keywords. The size of the tab is control is specified by the FLOC and FSIZE keywords used as part of the TABSET definition. The single option provided to the TAB keyword serves the dual purpose of being both the display name and the tab label name. The TABSET has a single option which is the fieldLabel of the TABSET.

The TABSET has a single option – tabsetDispatch.

When a user picks on a TAB, by default, it is dispatched to the application as the with the fieldLabel set to the name of the tab and the fieldValue as a 't'. With this option we use the fieldLabel defined with the TABSET keyword and the fieldValue as the tab name. In most cases you do not need to handle tab changes in your form dispatch code but when you do each dispatch method has its advantages.

Note: TABSETs cannot be nested.

GROUP

A visible box around other controls. As such, you give it a width, height and optional text. If width or height is 0, we draw the appropriate horizontal or vertical line. Normally the group text is static but you can change it at run-time by assigning a label to the group.

TEXT

Static text, defined in the form file with the keyword "TEXT". The optional second field (use double quotes if more then one word) is any text string that should appear in the field. An optional third field can be use to define a label for run-time control. In addition the label INFO can be used to define the field label and text width.

Mult-line text can be specified by using the FSIZE label with a the height greater then 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 then the width of the control).

There are three variations of the fillin control.

- single line text
- single line text with a drop-down (use POP keyword).
  The dropdown provides the ability to have pre-defined values for the user.
- multi-line text control. Use a FSIZE keyword to indicate field width and height.

INTFILLIN

Similar to a STRFILLIN except input data is checked to be an integer (numbers 0 to 9 and + and -). Use the LONGFILLIN keyword with two arguments; field width and string length.

It only supports variations 1 and 2 of STRFILLIN.

It also supports a minimum and maximum data verification. This can be done via the form file with the MIN and MAX keywords or at run-time via `axlFormSetFieldLimits`.

INTSLIDEBAR

This is a special version of the INTFILLIN, it provides an up/down control to the right of the field that allows the user to change the value using the mouse. You should use MIN/MAX settings to limit the allowed value.

REALFILLIN

Similar to INTFILLIN except supports floating point numbers. Edit checks are done to only allow [0 to 9 .+-]. If addition to min/max support you can also provide number of decimals via the DECIMAL keyword or at run-time via `axlFormSetDecimal`.

MENUBUTTON

Provides a button control. Buttons are stateless. The MENUBUTTON keyword takes two options; width and height.

A button has one option – multiline.

If button text cannot fit on one line wrap it. Otherwise text is centered and restricted to a single line.

A button can have a popup by inserting the "POP" label.

With no popup pressing the button dispatches a value of 1. If it is a button with a popup then the dispatch is the dispatch entry of the popup.

Standards:

- use "..." if button brings up a file browser
- append "..." to text of button if button brings up another window
- use these labels for:

  close - to Close dialog without

  done/ok - to store changes and close dialog

  cancel - to cancel dialog without making any changes

  help - The is reserved for cdsdoc help

  print - do not use (will get changed to Help).

CHECKLIST

Provides a check box control (on/off). Two variants are supported:

- a checkbox
- a radiobox

For both types the CHECKLIST control takes an argument for the text that should appear to the right of the checkbox.

A radiobox allows you to several checkboxes to be grouped together. The form package insures only one radiobox be set. To enable a radio groupping provide a common text string as a third argument to the CHECKLIST keyword. An idiosyncrasy of a radiobox is that you will be dispatched for both the field being unset and also for the field being set.

ENUM *(sometimes called combo box)*

Provides a dropdown to present the user a fixed set of choices. The dropdown can either be pre-defined in the form file via the POPUP keyword or at run-time with `axlFormBuildPopup`. Even if you choose to define the popup at run-time, you must provide a POPUP placeholder in the form file.

POPUP entries are in the form of display/dispatch pairs. Your setting and dispatching of this field must be via the dispatch item of the popup (you can always make both the same). This technique allows you to isolate what is displayed to the user from what your software uses. The special case of nil as a value to axlFormSetField will blank the control.

Two forms of ENUM field are supported, the default is single line always has the dropdown hidden until the user requests it. In this case only define the ENUMSET with the width parameter. A multi-line version is available where the drop-down is always displayed. To enable the multi-line version specify both the width and height in ENUMSET keyword.

Tip

FILLIN fields also offer ENUM capability, see below.

OPTIONS include (form file)

prettyprint - make text more read-able using upper/lower case.

ownerdrawn - provided to support color swatches next.

to subclass names. See axlSubclassFormPopup.

dispatchsame - Normally if user selects same entry that

is currently shown it will not dispatch.

LIST

A list box is a control that displays multiple items. If the list box is not large enough to display all the list box items at once, the list box provides the required horizontal or vertical scroll bar.

We support two list box types; single (default) and multi-selection. You define a multi-select box in form file with a "OPTIONS multiselect" List boxes have a width and height specified by the the second and third options to the LIST keyword. The first option to the LIST keyword is ignored and should always be an empty string ("").

List box options are:

SORT - alphabetical sort.

ALPHANUMSORT - takes in account trailing numbers so a NET2 appears before a NET10 in the list.

PRETTYPRINT - case is ignored and items are reformatted for readability.

Special APIs for list controls are: `axlFormListOptions`, `axlFormListDeleteAll`, `axlFormListSelect`, `axlFormListGetItem`, `axlFormListAddItem`, `axlFormListDeleteItem`, `axlFormListGetSelCount`, `axlFormListGetSelItems`, `axlFormListSelAll`.

For best performance in loading large lists consider passing a list of items to `axlFormSetField`.

THUMBNAIL

Provides a rectangular area for bitmaps or simple drawings. You must provide a FSIZE keyword to specify the area occuppied 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 hierarchial tree selector. See axlFormTreeViewSet.

GRID

This provides a simple spreadsheet like control. See axlFormGridDoc for more info.

COLOR

Provides a COLOR swatch. Can be used to indicate status (for example: red, yellow, green). The size of the color swatch is controlled by a width and height option the COLOR keyword.

Add the INFO_ONLY keyword to have a read-only color swatch. Without INFO_ONLY the color swatch provides CHECKBOX like functionality via its up/down appearance.

With COLOR swatches you can use predefine colors or Allegro database colors. See axlColorDoc.

TRACKBAR

Provides a slider bar for setting integer values. The TRACKBAR keyword takes both a width and height and the bar may be either horizontal or vertical.

It is important to set both a minimum and maximum integer value. This can be done from the form file with the MIN and MAX keywords or at run-time by `axlFormSetFieldLimits`.

PROGRESS

Provides a progress bar usually used to indicate status of time consuming operations. For setting options to the progress meter pass of list of 3 items to axlFormSetField which are (<step value> <number of steps> <initial position>). A subsequent nil passed to axlFormSetField will step the meter by the <*step value*>.

PROGRESS keyword provides for both a width and height of the bar. Bar should be horizontal.

You get information from the user using forms that support the following modes:

Table 10-1 Form Modes

Form Mode	Description
Blocking with no callback	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 `axlUIWBlock` after displaying the form. The user can close a form that has the standard *OK* or *Cancel* button. After *OK* or *Cancel* is selected, `axlUIBlock` returns allowing you to query field values using `axlFormGetField`. **Note:** Use this programming model only with simple forms.
Blocking with callback	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.
Callback with no blocking	Works like many native Allegro PCB Editor forms. The user can work with both the form and other parts of Allegro PCB Editor. 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 `axlEvent`.

Form Mode	Description
Options form	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 `axlMiniStatusLoad` for details.

Do not attempt to set the Button field (except *Done*, *Cancel* and *Help*), as it is designed to initiate actions. Consequently, having buttons in a form without a callback function registered renders those buttons useless.

Note: AXL-SKILL does not support the short fields and variable tiles which are part of the Allegro PCB Editor core form package.

You can set background and foreground color on many form fields. For more information, see axlFormColorize on page 587. For information on color specific to grids, see Using Grids on page 518.

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 `TELSKILL` 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.

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:
               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 [<tileLabel>]
          [TPANEL tileType]
          [{text_def}]
          [{group_def}]
          [{field_def}]
          [{button_def}]
          [{grid_def}]
          [{glex_def}]
          ENDTILE
tabset_def:
          TABSET [label]
          [OPTIONS tabsetOptions]
          FLOC x y
          FSIZE w h
          {tab_def}
          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
          FSIZE w h
          [OPTIONS INFO | HLINES | VLINES | USERSIZE ]
          [POP "<popupName>"]

          [GHEAD TOP|SIDE]
          [HEADSIZE h|w]
          [OPTION 3D|NUMBER]
          [POP "<popupName>"]
          [ENDGRID]
          ENDGRID

field_type:
          REALFILLIN w fieldLength |
          LONGFILLIN w fieldLength |
          STRFILLIN w fieldLength |
          INTSLIDEBAR w fieldLength |
          ENUMSET w [h] |
          CHECKLIST "display" ["radioLabel"] |
          LIST "" w h |
          TREEVIEW w h |
          COLOR w h |
          THUMBNAIL [<bitmapFile>|#<resource>] |
          PROGRESS w h
          TRACKBAR w h

field_options:
          [INFO_ONLY]
               -    Sets field to be read-only
```

```
[POP "<popupName>"]
     -    Assigns a popup with the field.
     -    A POPUP definition by the same name should exist.
     -    Supported by field_types: xxxFILLIN, INTSLIDEBAR, MENUBUTTON,and ENUMSET.
[MIN <value>]
[MAX <value>]
     -    Assigns a min and/or max value for the field.
     -    Both supported by field types: LONGFILLIN, INTSLIDEBAR, REALFILLIN.
     -    Value either an integer or floating point number.
```

```
          [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.

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]
               -    By default, tabsets dispatch individual tabs as seperate 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:
fy:
fh:
          -    Floating value between 0 and 1.0
```

Important

Follow these rules when using BNF format:

- `FILE_TYPE` line must always appear as the first line of the form file in the format shown.
- Form files must have a `.form` extension.
- There may only be one `FORM` in a form file.
- There must be one and only one `TILE` definition in a `FIXED` form file. <*`tileLabel`*> and `TPANEL` are not required.
- Unless otherwise noted, character limits are as follows:
  labels - 128
  title - 1024
  display - 128 except for *`xxx`*`FILLIN` 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 10-2 Controls - Resizing Options

Control	Resizing Options
REALFILLIN	width
LONGFILLIN	width
STRFILLIN	width
INTSLIDEBAR	width
ENUMSET	width

Table 10-2 Controls - Resizing Options, *continued*

Control	Resizing Options
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 INTSLIDEBAR 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*, *fy*, *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 l1 0 0 1 0.5
FLEX g1 0 0.5 1 0.5
FLEX l2 0 0 1 1
```

Form has a group (`g1`) containing a list (`l2`). These are at the bottom of another list (`l1`). Both lists share in any change of the form size. The second list (`l2`) is a member of the group container (`g1`), so it moves if the group moves (`0` for `y`) and it gets all of the group resizing (`h` is `1`).

Example 4

```
FLEX g1 1 1 0 0
FLEX l1 0 0 1 1
```

Form has a group (`g1`) with a list member (`l1`), but the list doesn't resize because the list is a member of the group which has `0:0` sizing. Though the list has `1:1` sizing, it never changes in size because its container never changes in size. Both the group and its member list move because the group has a `1:1` *`x/y`* factor.

Example 5

```
FLEX t1 0 0 1 1
FLEX l1 0 0 1 1
FLEX l2 0 0 1 1
```

Form is a tabset (`t1`) with 2 tabs. Each tab controls a list (`l1` and `l2`) that accommodates the maximum change in the form size.

➤ Use `axlFormTest(<`*`formname`*`>)` to experiment with your form.

Using Grids

Grids offer tabular support and the following features:

- Optional side and top headers
- Several data types on a per column basis: Text (info), Checkbox with optional text, Enum (Drop-drop) and Fillin (text box with built-in types: string, integer, and real.)
- Row and column indexing which is 1-based

Grids have the following limits:

- Maximum of 200 columns
- Maximum rows of 1,000,000
- Maximum field string length per column of 256 characters
- Column creation only at grid initialization time.

Form File Support for Grids

The following defines the form file structure relating to grids.

```
GRID
     Standard items
          FLOC -    x, y location
          FSIZE-    width and height including headers if used
          POP  -    Optional right button popup for body. Also requires application to set the
                         GEVENT_RIGHTPOPUP option.
     OPTIONS:
          INFO -    Entire grid is info-only even if it contains typeable fields
          HLINES-   Draw horizontal lines between columns
          VLINES-   Draw vertical lines between rows
          USERSIZE- Allow user to resize columns.
HEADERS (GHEAD)
     -    Specified within GRID section.
     -    TOP and SIDE header (only one per type allowed in a grid)
     HEADSIZE- Height (TOP) or width (SIDE) for the header.
     OPTIONS:
          3D   -    Display raised.
          NUMBER-   For side header, display row number if application does not provide text.
     POP  -    Optional right mouse button popup. One per header. Requires application to set
                         GEVENT_RIGHTPOPUP for the header.
```

Programming Support for Grids

The following Grid APIs are available:

`axlFormGridInsertCol`	Insert a column.
`axlFormGridInsertRows`	Insert one or more rows.
`axlFormGridDeleteRows`	Delete one or more rows.
`axlFormGridEvents`	Set grid events.
`axlFormGridOptions`	Miscellaneous grid options.
`axlFormGridNewCell`	Obtain structure for setting a cell.
`axlIsGridCellType`	Is item a cell data type.
`axlFormGridSetBatch`	For setting multiple cells.
`axlFormGridGetCell`	For getting cell data.
`axlFormGridBatch`	Used with `axlFormGridSetBatch`
`axlFormGridUpdate`	Update display after changes.
`make_formGridCol`	For defstruct `formGridCol`
`copy_formGridCol`	For defstruct `formGridCol`

In addition, the following standard form APIs may be used:

`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 *change* event. Adding, deleting, or right mouse event reset restore.

Data Structures

`r_cell`	User data type for cell update (see axlFormGridNewCell on page 605)
`r_formGridCol`	Defstruct to describe column (see axlFormGridInsertCol on page 599)

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.
`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 594.)

By default, you create a grid with the 'rowselect enabled which is typically appropriate for a multi-column table.

Using Scripting with Grid Controls

Unlike most other form controls where the programmer needs no concern over scripting, grid programmers should address scripting. By default, the grid uses the event type and row/column number for scripting. Depending on your application, this may create scripts that do not replay given different starting data. Grids support assigning script labels to rows, to columns, and on a per cell basis.

You label by setting the *scriptLabel* attribute from the application code with the axlFormGridInsertCol function for a column or the axlFormGridNewCell function for a row, column, or per cell basis. You can also change this dynamically. Note that (row=0, col=*n*) sets the *scriptLabel* for the column using axlFormGridNewCell and (row=*n*, col=0) allows setting for row script labels.

The grid script line format extends upon the standard form scripting as shown:

```
FORM <formname> [tileLabel] <fieldLabel> <event> <glabel> [<value>]
```

where

```
FORM <formname> [tileLabel] <fieldLabel>
-    standard form script form fieldLabel is the grid label
<event> is the grid event. Grid events include:
     rowselect:= GEVENT_ROWSELECT
     cellselect:= GEVENT_CELLSELECT
     change:= GEVENT_CELLCHANGE
     rpopup:= GEVENT_RIGHTPOPUP
     rprepopup:= GEVENT_RIGHTPOPUPPRE
     lprepopup:= 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>
lprepopup<glabel:=cell>
```

The *glabel* has several format options depending on the event:

row	If the row has a *scriptLabel*, it is used, otherwise the row number is used.
cell	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.

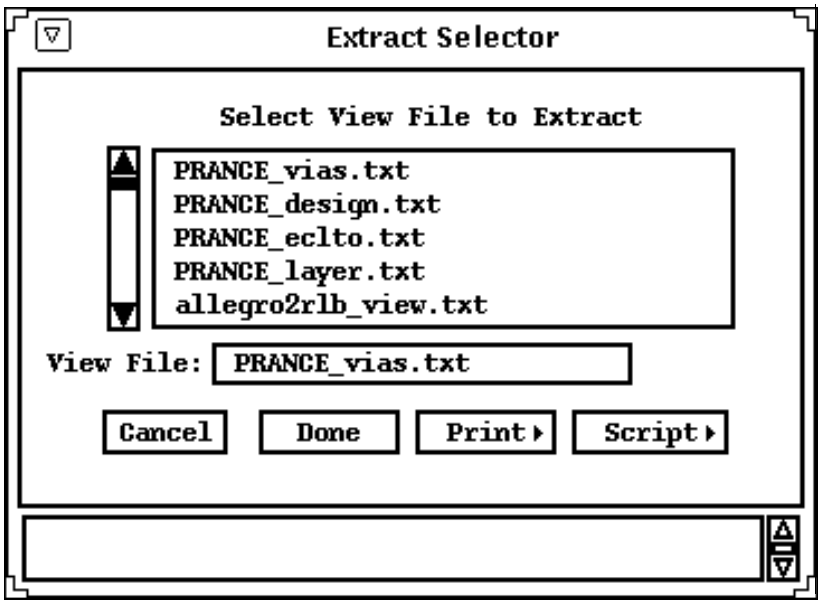
See Using Grids on page 518 for a grid overview.

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
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.

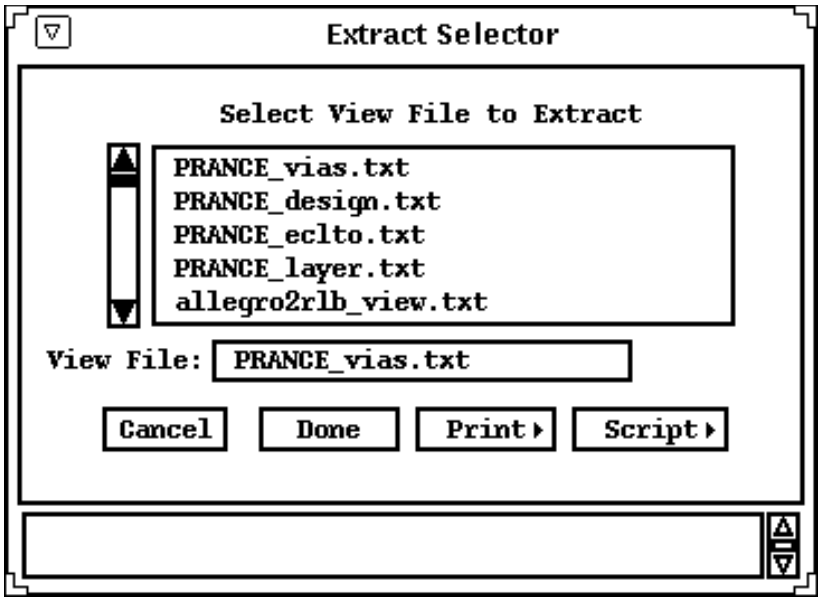
  The form file has `FIELD` definitions for the selection list, the *View File* field, and each of the buttons (*Cancel*, *Done*, *Print* and *Script*).

```
; myExtractViews.il
;             -- Displays a form with a selection list of
;                  the available extract definition files
;             -- Lets the user select any of the files on
;                  the list as the "View file"
;             -- Starts Allegro extract process with the
;                  user-selected View file when
;                 the user picks Done from the form.

; Function to extract user selected view to the output file.
(defun myExtractViews (viewFile outFile)
     axlExtractToFile( viewFile outFile)
); defun myExtractViews
; Function to start the view extraction
(defun _extract ()
 myExtractViews(
buildString(list(cadr(parseString(
     axlGetVariable("TEXTPATH"))) selectedFile) "/")
               "myextract.dat")
); defun _extract
; Form callback function to respond
(defun _formAction (form)
     (case form->curField
          ("done"
               (axlFormClose form)
               (axlCancelEnterFun)
               (_extract)
               t)
          ("cancel"
               (axlFormClose form)
               (axlCancelEnterFun)
               nil)
          ("view_file"
               (if form->curValue
                    (progn
                         ; Accept user input only if on list
                         if(member( form->curValue fileList)
                             then axlFormSetField( form
                                  "view_file" form->curValue)
                              else axlFormRestoreField(
                                       form "view_file"))))
               t)
          ("file_list"
               (axlFormSetField form "view_file"
                   form->curValue)
               selectedFile = form->curValue
               t)); case
); defun _formAction
; User-callable function to set up and
;        display the Extract Selector form
(defun myExtract ()
     fileList = (cdr (cdr (getDirFiles
     cadr( parseString( axlGetVariable("TEXTPATH"))))))
     form = axlFormCreate( (gensym)
          "extract_selector.form" '("E" "OUTER")
              '_formAction t)
 axlFormTitle( form "Extract Selector")
 axlFormSetField( form "view_file" (car fileList))
 selectedFile = (car fileList)
 foreach( fileName fileList
          axlFormSetField( form "file_list" fileName))
      axlFormDisplay( form)
); defun myExtract
```

- Creates a form named `form` with the callback function `_formAction` that analyzes user action stored in *`form->curField`* and responds appropriately.

- Loads the example AXL program shown.
- Enters the command `myExtract()`.

  SKILL displays the **Extract Selector** form, as specified in the form file `extract_selector.form` that this code created when it first loaded. This is a non-blocking form—you can enter other SKILL and 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.
`"view_file"`	The user selected the *View File* field, possibly typed an entry, and pressed *Return*. Sets the *`view file`* name to the current

	value of the *View File* field, letting the user type in a name. Name must be a name on the list displayed.
`"file_list"`	The user picked a name from the displayed list of view file names. Name picked is *`form->curValue`*, and the program sets *`selectedFile`* (the name of the currently selected extract file) to the new value, and displays it in the *View File* field.

The *Print* and *Script* buttons have pop-ups that call predefined 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"
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")
```

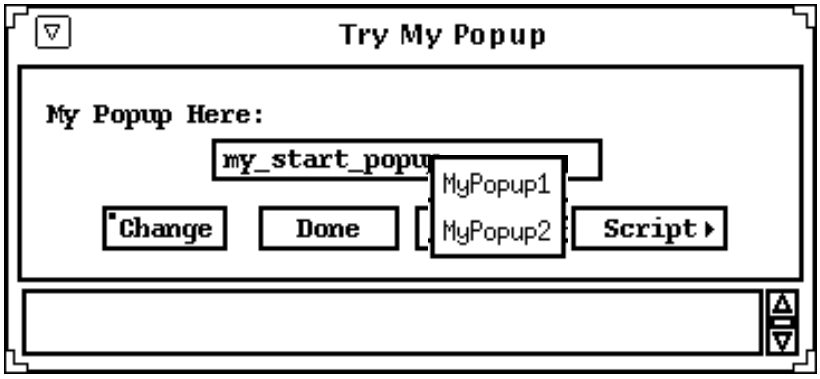
A list of strings, which means that each string represents both the display and dispatch values of that popup selection.

```
; 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"
                                        "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
```

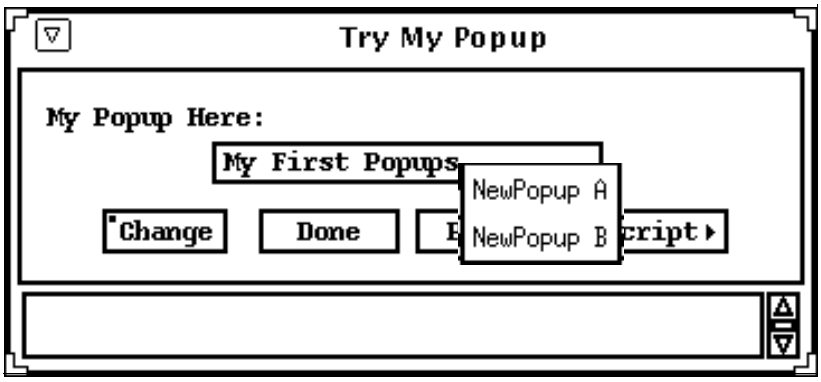
Sets the field *my_popup* to the value selected by the user and prints it.

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*.

   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.

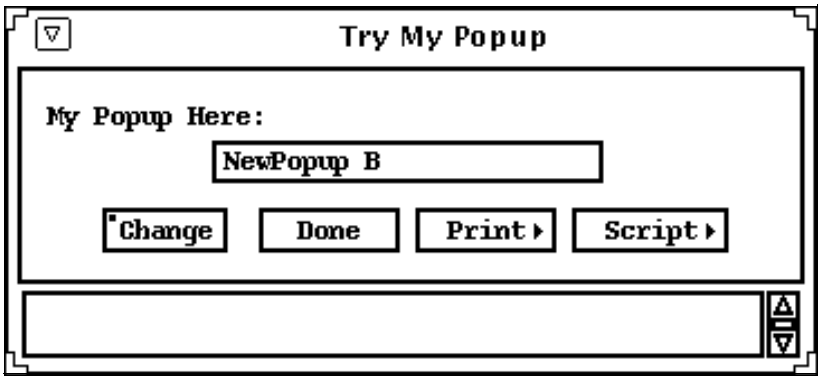


4. Make a selection.

   If, for example, you selected *NewPopup B*, the program prints the following on the SKILL command line:

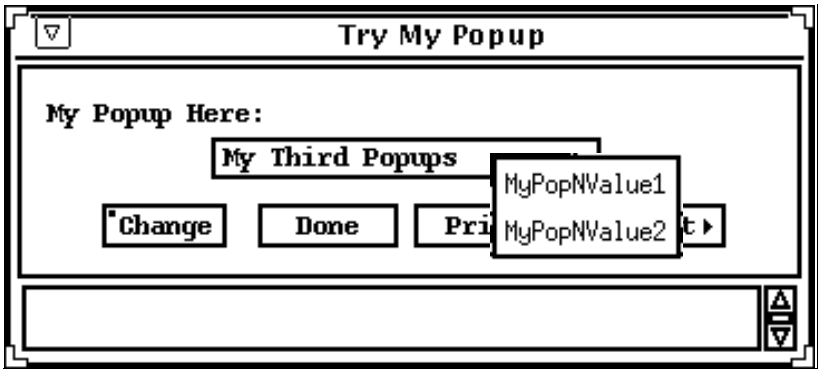
```
Got my_popup event: form->curValue mynewpopup_b
```

The following form is displayed.



5. Click *Change*.

   The program displays the third set of pop-ups.



AXL-SKILL Form Interface Functions

This section lists the form interface functions.

axlFormBNFDoc

This is the BNF grammer for the Forms Specification Language. New options and field types are added every release. Form files are always upwards compatible but may NOT be backwards compatible if you take advantage of a new feature. Thus, a form file created in 12.0 Allegro works in 13.0 Allegro. However, if you take advantage of the TAB control (13.0) or the RIGHT justification of TEXT (13.5), you will have a form file that will not function with 12.0 of Allegro.

The following outlines the conventions used in the grammar:

[]	Optional
{}	May repeat one or more times.
<>	Supplied by user.
|	Choose one or the other.
:	Definition of a token.
CAPS	Items in caps are keywords (note form parser is case insensitive)
(#)	Note: See number at end of this documentation.

BNF

form

```
FILE_TYPE=FORM_DEFN VERSION=2 (1)
FORM [form_options]     (3)
formtype
PORT w h
HEADER "text"
form_header
{tile_def}
ENDFORM
```

formtype FIXED | VARIABLE

- FIXED forms have a one unlabled 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 BUTTONs. 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 toolwindow which is a floating toolbar. It is typically used as a narrow temp window to display readouts.

[FIXED_FONT]
- by default forms use a variable width font, this sets this form to use a fixed font. Allegro PCB Editor uses mostly variable width while Allegro PCB SI and SigXplorer use fixed width fonts.

[AUTOGREYTEXT]
- when a fillin or enum control is greyed, grey static text to the left of it.

[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.

tile_def

```
TILE [<tileLabel>]      (4)
[TPANEL tileType]
[{text_def}]
[{group_def}]
[{list_def}]
[{field_def}]
[{button_def}]
[{grid_def}]
[{flex_def}]
ENDTILE
```

tabset_def

```
TABSET [label]
[OPTIONS tabsetOptions]
FLOC x y
FSIZE w h
{tab_def}
ENDTABSET
```

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)
field_type
field_options
ENDFIELD
```

button_def

```
FIELD label
FLOC x y
[FSIZE w h]        (8)
MENUBUTTON "display" w h
button_options
ENDFIELD
```

grid_def

```
GRID fieldName
FLOC x y
FSIZE w h              (8)
[OPTIONS INFO | HLINES | VLINES | USERSIZE ]
[POP "<popupName>"]

[GHEAD TOP|SIDE]
[HEADSIZE h|w]
[OPTION 3D|NUMBER]
[POP "<popupName>"]
[ENDGREADH]
ENDGRID
```

field_type

```
REALFILLIN w fieldLength |
LONGFILLIN w fieldLength |
STRFILLIN w fieldLength |
INTSLIDEBAR w fieldLength |
ENUMSET w [h] |                (11)
CHECKLIST "display" ["radioLabel"] |
LIST "" w h |
TREEVIEW w h |
COLOR w h |
THUMBNAIL [<bitmapFile>|#<resource>] |
PROGRESS w h
TRACKBAR w h
```

field_options

```
[INFO_ONLY]
    - sets field to be read only.

[POP "<popupName>"]
    - assigns a popup with the field.
    - a POPUP definition by the same name should exist.
    - supported by field_types: xxxFILLIN, INTSLIDEBAR, MENUBUTTON,
    ENUMSET

[MIN  <value>]
[MAX  <value>]
```

- assigns a min and/or max value that field might have.
- both supported by field_types: LONGFILLIN, INTSLIDEBAR, REALFILLIN.
- value by either by an interger 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]
- alphanumberic 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`.

[OPTIONS space]

- string type only

- preserves leading and trailing white space. By default this is stripped.

list_options

[OPTIONS sort|alphanumsort|prettyprint|multiselect]
sort - convertion 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 then one item (follows Microsoft selection model).

x
y
w
h

- display geometry (integers).
- all field, group and text locations are relative to the start of the tile they belong or to the start of the form in the case of FIXED forms.
- x & h are in CHARHEIGHT/2 units.
- y & w are in CHARWIDTH units.

button_options

[MULTILINE]
- wraps button text to multiple lines if text string is too long for a single line.

dispatch

- string that is dispatched to the code.

display

- string that is shown to user

bitmapFile

- name of a bmp file. Assumes can be found via BITMAPPATH

resource

- integer resource id (bitmap must be bound in executable via the resource file). '#' indicates it is a resource id.
- not support in AXL forms.

fieldLength

- maximum width of field. Field will scroll if larger then field display width.

label

- named used to access field from code. All fields should have unique names.
- should use lower case.

messageLabel

- name used to allow code to refer to messages.
- case insensitve

messagePriority

- message priority 0 - info (not in journal file), 1 - info, 2 - warning, 3 - error, 4 fatal (display in message box).

radioLabel

- named used to associate several CHECKLIST fields as a radio button set. All check fields should be given the same radioLabel.
- should use lower case.

textOptions

RIGHT | CENTER | BORDER | BOLD | UNDERLINE]
- TEXT/INFO field type.
- text justification, default is left.
- BORDER: draw border around text.

[STRETCH]
- THUMBNAIL field type.
- stretch bitmap to fit thumbnail rectangle, default is center bitmap.

[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 tabsets dispatch individual tabs as seperate events. This is not always convenient for certain programming styles. This changes the dispatch mode to be upon the tabset where a selection of a tab causes the event field=tabsetLabel value=tabLabel.
The default is:
field=tabLabel value=t
Script record/replay remains based upon tab in either mode.

tileLabel

- name used to allow code to refer to this tile.
- should use lower case.
- only applies to VARIABLE forms.
- not support with AXL forms.

tileType [0|1|2]

- 0 top tile, 1 scroll tile, 2 bottom tile.
- only applies to VARIABLE FORMS.
- region where tile will be instantiated. Forms have 3 regions top, bottom and scroll (middle).
- not support with AXL forms.

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.

- 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

Note:

- FILE_TYPE line must always appear as the first line of form file in format shown.
- Form files must have a .form extension.
- There may only be one FORM in a form file.
- There must be one and only one TILE defintion in a FIXED form file. <tileLabel> and TPANEL are not required.
- Unless otherwise noted limits are as follows:
  labels - 128
  title - 1024
  display - 128 except for xxxFILLIN types which are 1024
- Additional items may appear in existing form files (FGROUP) but they are obsolete and are ignored by the form parser. REALMIN & REALMAX are obsolete and replaced by MIN and MAX respectively. They will still be supported and are mapped to MIN and MAX.
- For grid_def two headers (side and top) are maximum.
  SIZE - most controls determine the size from the text string. You are required to provide FSIZE for GROUP, GRID, TREEVIEW and LIST controls. For TEXT controls if FSIZE is provided after it overrides the width calculated by the text length and if present the INFO width. If the INFO line appears you should put the FSIZE line after it.
- Both TEXT and GROUP support optional label on their definition line. This was added as a convience 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 then 1 then the dropdown is always visible (Microsoft SIMPLE drop-down). You only want to use this feature in forms that can afford the space consumed by the drop-down.

The forming syntaxes are NOT supported by the formeditor.

This following syntax is supported and may be placed anywhere in the form file to support conditional processing of the form file:

#ifdef <variable>|
{ }

{ #elseif <variable>
}

{ #else
{ } }

axlFormCallback

```
formCallback(
          [r_form]
          )
==> t
```

Description

This is not a function but documents the callback interface for form interaction between a user and Skill code. The Skill program author provides this function.

When the user changes a field in a form the Allegro form processor calls the procedure you specified as the `g_formAction` argument in `axlFormCreate` when you created that form. The form attribute `curField` specifies the name of the field that changed. The form attribute `curValue` specifies the current value of the field (after the user changed it). If you set `g_stringOption` to `t` in your call to `axlFormCreate` when you created that form, then `curValue` is a string. If `g_stringOption` was `nil` (the default), then `curValue` is the type you specified for that field in the form file.

Note: The term `formCallback` used in the title of this callback procedure description is a dummy name. The callback function name must match the name or symbol name you used as the `g_formAction` argument in `axlFormCreate` when you created the form.

If you specify the callback name (`g_formAction`) as a string in your call to `axlFormCreate`, SKILL calls that function with no arguments. If you specify `g_formAction` as a symbol, then SKILL calls that function with the form handle as its single argument.

The callback must call `axlFormClose` to close the form and to continue in the main application code if form mode is blocking.

All form infomation is provided by the `r_form` argument which is a form data type. Applications can extend the data stored on this type by adding their own attributes. Please capitalize the first letter of the attribute name to avoid conflicts with future additions by Cadence to this structure. Tables 1 and 2 show the available field types and how they impact the `r_form` data type.

Table 1

Form Field Types:

Type — What the field is commenly known to the user.

Keyword — How the field is declared in the form file (see `axlFormBNF`Doc).

curValue — The data type seen in the form dispatch and `axlFormGetField` (see `axlFormCallback`).

curValueInt — If `curValue` 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: axlFormTreeViewSet
Text	INFO (4)	N/A	N/A
Graphics	THUMBNAIL (5)	N/A	N/A
Trackbar	TRACKBAR	integer	integer
Grid	GRID	See: axlFormGridDoc	

Note:

- What distinguishes between a radio button and check box is that a radios buttons are a group of check boxes where only one can be set. To related several checkboxes as radio buttons use supply the same label name as the third field (groupLabel) in the form file description:

  ```
  CHECKLIST <fieldLabel> <groupLabel>
  ```

  When a user sets a radio button the button be unset will dispatch to the app's callback with a value of nil.

- Enum will only set `curValueInt` on dispatch when the dispatch value of their popup uses an integer. Otherwise this field is `nil`.

- Tabs can dispatch in two methods:

  - default when a tab is selected your dispatcher receives the tab name in the `curField` and `curValue` is `t`.

  - If `OPTIONS tabsetDispatch` is set in the TABSET of the form file then when a tab is selected your app dispater receives the TABSET as the `curField` and the `curValue` being the name of the TAB that was selected.

- INFO fields can be static where the text is declared in the form file or dynamic where you can set the text via the application at run-time. To achieve dynamic access enter the following in the form file:

  ```
  TEXT "<optional initial text>"
  INFO <fieldLabel>
  ```

  ... reset of TEXT section ...

- Thumbnails support three methods:

  - Static bitmap declared via form file.

  - Bitmaps that can by changed by the application at run-time.

  - Basic drawing canvas (see `axlGRPDoc`).

- Buttons are stateless. The application cannot set the button to the depressed state. You can only use `axlFormSetField` to change the text in the button. Several button fieldLabels are reserved. Use them only as described:

Done or *Ok*	Do action and close form.
Cancel	Cancel changes and close form.
Print	Print form; do not use.
Help	Call cdsdoc for help about form. Do not use.

Table 2

Attribute Name	Set?	Type*	Description
curField	no	string	Name of form field (control) that just changed.
curValue	no	See->	Value dependant upon field type (2).
curValueInt	no	See->	Value dependant upon field type (2).
doneState	no	int	0 = action; 1 = done; 2 = cancel; 3 = abort (1)
form	no	string	Name of this form (form file name).
isChanged	no	`t` / `nil`	`t` = user has changed one or more fields.
isValueString	no	`t` / `nil`	`t` all field values are strings. `nil` one or more fields are not strings.
objType	no	string	Type of object; in this case `form`.
type	no	string	Always `fixed`.
fields	no	list of strings	All fields in the form (3).
infos	no	list of strings	All info. fields in form (3).
event	no	symbol	List, tree and grid control only. See `axlFormGridDoc` for grid info.
row	no	integer	Grid control only.
col	no	integer	Grid control only.
treeViewSelState	no	integer	Tree control only.

Note:

- The `doneState` shows 0 for most actions. If a button with *Done* or *Ok* is pushed, then the done state is set. A button with the *Cancel* label sets the cancel state.

In either the Done or Cancel state, you need to close the form with `axlFormClose`. If the abort state is set, the form closes even if you do not issue an `axlFormClose`.

- Data type is dependant upon the field type, see Table 1.
- The difference between the fields and infos list is that items appearing in the infos list are static text strings the program can change at run-time. All other labels appear in the fields list and are reflect they can be changed by the user (even buttons, tabs, greyed and hidden fields).
- Event for list box is `t` if item is selected, `nil` if deselected. This is always `t` for single select list box while the multi-select option can have both states.
- Event and treeViewSelState for a tree control see `axlFormTreeViewAddItem`.

Arguments

r_form	Form dbid.

Value Returned

`t`	Always returns `t`.

Examples

See `axlFormCreate` and `axlFormBuildPopup` examples.

axlFormCreate

```
axlFormCreate(
     s_formHandle
     t_formfile/(t_formName t_contents)
     [lt_placement]
     g_formAction
     g_nonBlock
     [g_stringOption]
)
⇒ r_form/nil
```

Description

Creates a form structure based on the form descriptive file *t_formfile*. This call only supports forms of type "fixed" and will fail if *t_formfile* contains any variable tiles. This function does not display the form. Use axlFormDisplay to display a form.

An alternative interface is supported that allows embedding the contents of the form file with the skill code. Instead of passing the external form file name provide the name (t_formName) for scripting purposes and 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/axlform.il file.

Two things to remember when creating this form content string:

- Each form file line must end with a - \n

  Example:

  ```
  FILE_TYPE=FORM_DEFN VERSION=2\n
  ```

- Any embedded quotes must be backslashed

  Example:

  ```
  MENUBUTTON \"Ok\" 10 3\n
  ```

Note: If *s_formHandle* is an existing *r_form*, then axlFormCreate does not create a new form, but simply exposes and displays the existing form, *s_formHandle*, and returns nil.

Arguments

s_formHandle	Global SKILL symbol used to reference form.

Note: Do not use the same symbol to reference different form instances.

t_formfile

Filename of the form file to be used to define this form. `axlFormCreate` uses the Allegro PCB Editor environment variable, `FORMPATH`, to find the file, if *`t_formfile`* is not a full path. The filename, by convention, should use the `.form` extension.

Alternative interface to embed form file into Skill code:

- `t_formName`: Name of form (used for scripting)
- `t_contents`: contents of form file

lt_placement

Form placement. Allegro PCB Editor uses its default placement if this argument is `nil`. See Window Placement on page 425

g_formAction Specifies the SKILL commands (callbacks) to execute after every field change (Note that this is very different from Opus forms). You can set this to one of the formats shown:

g_formAction Options

Option	Description
t_callback	String representation of the SKILL command to be executed.
s_callback	Symbol of the SKILL function to be called (passes the *r_form* returned from axlFormCreate as its only parameter.)
nil	axlFormDisplay blocks until the user closes the form.You must place a *Done* button (field name done) and optionally a *Cancel* button (field name cancel) in the form for *g_formAction* to function properly. The user can access all of the fields and values using the *r_form* user type.

g_nonBlock If *g_nonBlock* is t, the form runs in non-blocking mode. In blocking mode (the default), axlFormDisplay blocks until the user closes the form. Blocking is an easier programming mode but might not be appropriate for your application. If the callback (*g_formAction)* is nil, then axlFormDisplay ignores *g_nonBlock*, and the form runs in blocking mode.

Use of blocking mode blocks the progress of the SKILL code, but does not prevent other Allegro PCB Editor events from occurring. For example, if blocked, users can start the *Add Line* command from Allegro PCB Editor menus.

g_stringOption If t, the form returns and accepts all values as strings. By default, it returns and accepts values in the format declared in the form file.

Value Returned

r_form *dbid* of form created.

nil No form created.

Example

See AXL Forms: Example 1 on page 525.

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 525:

```
(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 525.

```
axlFormDisplay( form)
```

axlFormBuildPopup

```
axlFormBuildPopup(
     r_form
     t_field
     l_pairs

)

⇒ t/nil
```

Description

Builds a pop-up for field *t_field* in the open form *r_form*. Field must be an enumerated list type field. axlFormBuildPopup replaces the existing pop-up attached to the field. The *l_pairs* argument is a list of display and dispatch string pairs.

Arguments

r_form	Form *dbid*.
t_field	Name of form field.
l_pairs	Must be one of the formats described. Each list object defines a single popup entry. Options can be 1 or 2 additional list options that are S_color/ x_color for enum field types with color; bold or underline for bold or underlined items.

l_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 *form->curValue* 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 *form –curValue* 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")`

Option	Description	Example

Optional field	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`).	You can't mix this color type in a single popup.
		`'(("Green" 1 green) ("Red" 2 red) ("Yellow" 3 yellow))`
		`'(("Top" "top" 2) ("Gnd "gnd" 4) ("Bottom" "btm" 18))`
		If instead of a color or Allegro color number, you provide a `nil`, then that popup entry will not have a color swatch.
		`'("(None" 0 nil) ("Green" 1 green) ("Red" 2 red) ("Yellow" 3 yellow))`
		Font type of bold or underline can be specified via:
		`'(("Top" "top" bold) ("Gnd "gnd" underline) ("Bottom" "btm"))`
		When font type is combined with color it looks like:
		`'(("Top" "top" "Green" bold) ("Gnd "gnd" "Red" underline)`

Notes:

- Allows a maximum of 256 pop-up entries in one pop-up.
- All entries in an *`l_pairs`* argument must be the same type of format. That is, you cannot have a list containing both display/dispatch strings and display/enum types, or display/dispatch and single-string entries.

Value Returned

`t`	Field set.
`nil`	Field not set.

Example

See AXL Forms: Example 2 on page 529.

axlFormGetField

```
axlFormGetField(
     r_form
     t_field
)
⇒ g_value/nil
```

Description

Gets the value of *t_field* in the open form *r_form*. The value is a string if *g_stringOption* was set in axlFormCreate. Otherwise the value is in the field type declared in the form file.

Arguments

r_form	Form *dbid*.
t_field	Name of field.

Value Returned

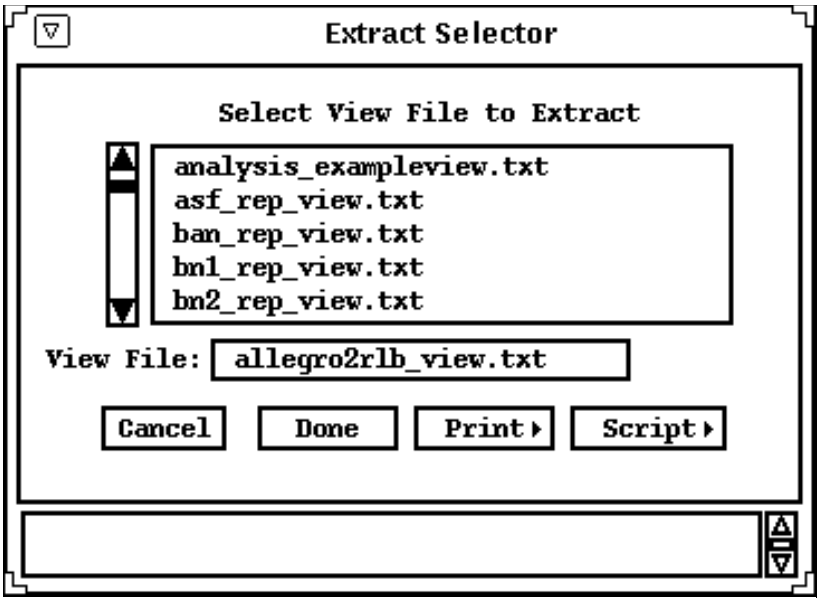
g_value	Current value of the field.
nil	Field does not exist, or false if boolean field such as check box or radio button.

Example

1. Load the example code given in AXL Forms: Example 1 on page 525.
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.

`allegro2rlb_view.txt` is entered.




```
axlFormGetField( form "view_file")
     ⇒ "allegro2rlb_view.txt"
```

Examines the value of "*view_file*" .

axlFormListDeleteAll

```
axlFormListDeleteAll(
     r_form
     t_field

)

⇒ t/nil
```

Description

Deletes all the items from the form list field, *t_field.*. Use `axlFormListDeleteAll` to clear an entire list field to update it using `axlFormSetField`, then display it using `axlFormSetField` on the field with a `nil` field value.

Arguments

r_form	Form *dbid*.
t_field	Name of field.

Value Returned

`t`	All items deleted properly.
`nil`	All items not deleted.

Examples

In this example you do the following:

1. Use the `axlFormCreate` examples to create and display the Extract Selector dialog box shown in Figure 10-1 on page 565.
2. On the SKILL command line, enter:

   ```
   axlFormListDeleteAll(form "file_list")
        ==> nil
   ```

   The list is removed from the dialog box as shown in Figure 10-2 on page 565.
3. On the SKILL command line, enter:

```
axlFormSetField(form "file_list" "fu")
axlFormSetField(form "file_list" "bar")
axlFormSetField(form "file_list" nil)
==> t
```

The Extract Selector dialog box is displayed with new list as shown in Figure 10-3 on page 566.

Figure 10-1 Extract Selector Dialog Box

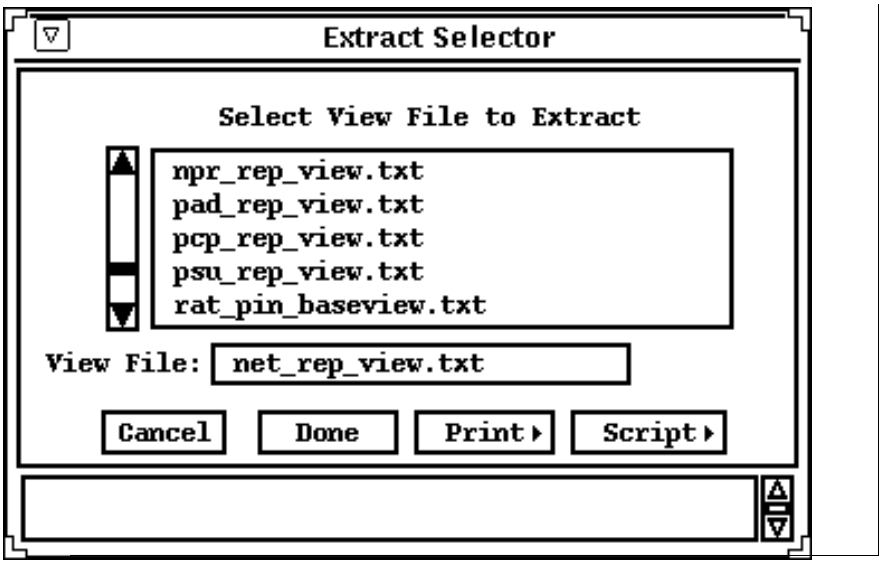


Figure 10-2 Extract Selector Dialog Box - List removed

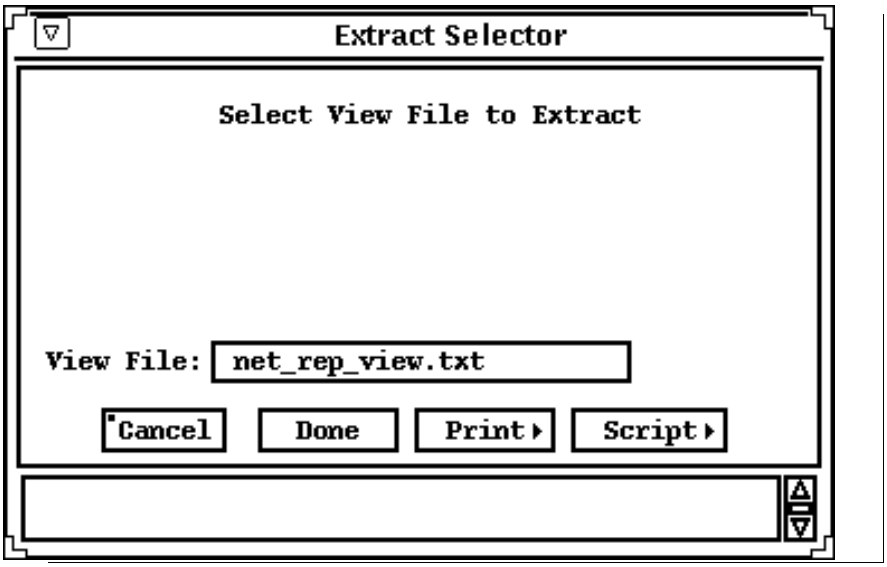
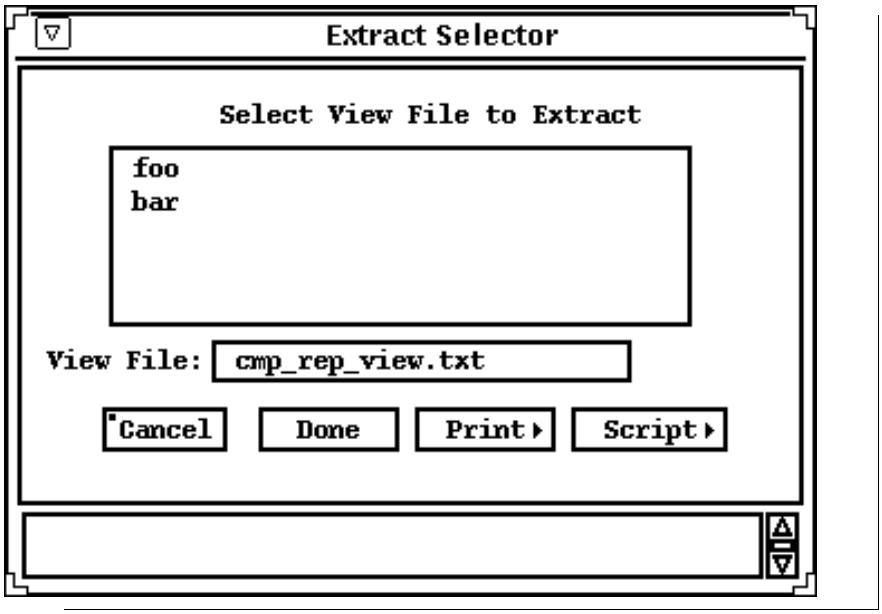


Figure 10-3 The Extract Selector dialog box - Displayed with a new list



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

`r_form`	Form id
`t_field`	Name of field.
`t_listItem``/nil`	String of item in the list. Send `nil` to deselect any selected item and set list back to top.

Value Returned

`t`	Highlights item. Arguments are valid.
`nil`	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

r_form	Form *dbid*
g_callback	Specifies the SKILL command(s) (callback(s)) to be executed whenever a new field is activated. The setting can be one of two formats: *t_callback* : the string representation of the SKILL command(s) to be executed *s_callback* : the symbol of the SKILL function to be called (the function is passed the *r_form* returned from `axlFormCreate` as its only parameter).

See Also

axlFormBNFDoc and axlFormCreate

Value Returned

`t`	Field set to desired value.
`nil`	Field not set to the desired value due to invalid arguments.

EXAMPLE

```
form = axlFormCreate( MyForm
     "extract_selector.form" '("E" "OUTER")
     '_formAction t)
axlFormSetEventAction( form '_formEventAction)
```

axlFormSetField

```
axlFormSetField(
     r_form
     t_field
     g_value/nil
)
⇒ t/nil
```

Description

Sets *t_field* to value *g_value* in open form *r_form*. Must pass the correct type, matching the entry in the form value or string type. Value type is dependent upon type of field type. 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:

`s_colorSymbol`	Set field to predefined color
`x_number`	Set field to product color
`t` or `nil`	Depress or raise field
`l_both`	A list allows setting both check and value; pass a list of the color set

s_colorSymbol may be black, white, red, green, yellow.

x_number is an integer between 1 and 24 with 0 being background.

Arguments

r_form	Form *dbid*.

t_field	Name of field.
g_value	Desired value of field.

Value Returned

t	Field set to desired value.
nil	Field not set to the desired value due to invalid arguments.

Examples

See AXL Forms: Example 1 on page 525.

```
axlFormSetField( form "file_list" fileName)
```

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)
;; 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)
```

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

r_form	Form *dbid*.
t_field	Name of field.
t_value	Desired value of field.

Value Returned

t	Field was set to desired value.
nil	Field not set to desired value due to invalid arguments.

Example

See the use of axlFormSetField in the "AXL Forms: Example 1" on page 525.

```
axlFormSetInfo( form "file_list" fileName)
```

axlFormTest

```
axlFormTest(
                    t_formName
          ) r_form/nil
```

Description

This is a development function for test purposes. Given a form file name this opens a form file to check for placement of controls. If form uses standard button names (for example, *ok, done, close, cancel*), you can close it be 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 succesfully 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

r_form	Form *dbid*.
t_field	Name of field.

Value Returned

`t`	Field restored.
`nil`	Field not restored and may not exist.

Example

See "AXL Forms: Example 1" on page 525 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

r_form	Form *dbid*.
t_title	String to be used for new form title

Value Returned

t	Changed form title.
nil	No form title changed.

Example

See "AXL Forms: Example 1" on page 525.

```
axlFormTitle( form "Extract Selector")
```

axlIsFormType

```
axlIsFormType(
     g_form
)
⇒ t/nil
```

Description

Tests if argument *g_form* is a form *dbid*.

Arguments

g_form	*dbid* of object to test.

Value Returned

t	*r_form* is the *dbid* of a form.
nil	*r_form* is not the *dbid* of a form.

Example

```
form = axlFormCreate( (gensym)
     "extract_selector.form" '("E" "OUTER")
     '_formAction t)
if( axlIsFormType(form)
     then (print "Created form successfully.")
     else (print "Error! Could not create form."))
```

Checks that the form you create is truly a form.

axlFormSetFieldVisible

```
axlFormSetFieldVisible(
     r_form
     t_field
     x_value
)
⇒ t/nil
```

Description

Sets a form field to visible or invisible.

Arguments

r_form	Form id.
t_field	Form field name (string).
x_value	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

r_form	Form id.
t_field	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 10-3 on page 582 and Table 10-4 on page 584 show the available field types and how they impact the *r_form* data type.

Table 10-3 on page 582 describes Form Field Types using the following:

Type	What the user calls the field
Keyword	What the form file calls the field

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 10-3 Form Field Types

Type	Keyword	curValue	curValueInt
Button	`MENUBUTTON`	dispatch action only (t)	1
Check Box	`CHECKLIST`	`t/nil`	0 or 1
Radio Button	`CHECKLIST`	`t/nil`	0 or 1
Long (integer)	`INTFILLIN`	integer number	Integer
Real (float)	`REALFILLIN`	float number	n/a
String	`STRFILLIN`	string	n/a
Enum (popup)	`ENUMSET`	string	Possible integer[1]
List	`LIST`	string	Offset from start of list (`0` = first entry).
Color well	`COLOR`	`t/nil`	`1` or `0`
Tab	`TABSET/TAB`	string or `t`	n/a or `1`/`0`
Tree	`TREEVIEW`	string	see `axlFormTreeViewSet`
Text	`INFO`	n/a	n/a
Graphics	`THUMBNAIL`	n/a	n/a
GRID	`GRID`	see Using Grids on page 518	

1. Integer if the dispatch value of the pop-up is an integer.

Notes:

- What distinguishes between a radio button and a check box is that radio buttons are a group of boxes where only one can be set. To relate several checkboxes as radio buttons, supply the same label name as the third field (groupLabel) in the form file description:

```
CHECKLIST <fieldLabel> <groupLabel>
```

When a user sets a radio button, the button being unset will dispatch to the application's callback with a value of nil.

- Enum will only set curValueInt on dispatch when their dispatch value of their popup uses an integer. Otherwise this field is `nil`.
- Tabs can dispatch in two methods:
  - Default when a tab is selected, your dispatcher receives the tab name in the curField and curValue is t.
  - If "OPTIONS tabsetDispatch" is set in the TABSET of the form file, then when a tab is selected your application dispatcher receives the TABSET as the curField and the curValue being the name of the TAB that was selected.
- INFO fields can be static where the text is declared in the form file or dynamic where you can set the text via the application at run-time. To achieve dynamic access, enter the following in the form file:

```
TEXT "<optional initial text>"
INFO <fieldLabel>
... reset of TEXT section ...
```

- Thumbnails support the following methods:
  - static bitmap declared via the form file
  - bitmaps that can be changed by the application at run-time
  - basic drawing canvas -- see Chapter 11, "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 10-4 Form Attributes

Attribute Name	Set?	Type*	Description
curField	no	string	Name of form field just changed
curValue	no	See -->	Depends on value of *curField* (`string`, `int`, `float`, `boolean`)
curValueInt	no	See -->	Depends on value of *curField* field
doneState	no	int	`0` = action; `1` = done; `2` = cancel; `3` = abort
form	no	string	Name of this form
isChanged	no	`t/nil`	`t` = user has changed one or more fields in form.
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	Form type, always `"fixed"`
fields	no	list of strings	All fields in the form.
infos	no	list of strings	All info fields in the form.
event	no	symbol	Grid control only -- see Using Grids on page 518
row	no	integer	Grid control only
col	no	integer	Grid control only
treeViewSelState	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 dependant on the field type. See Table 10-3 on page 582 for more information on Form Field Types.
- The infos list is different from the fields list. The infos list comprises static text strings that the program can change at run-time. The fields list comprises all other labels which can be changed by the user including even those on buttons and tabs, greyed and hidden fields.

Arguments

r_form	Form *dbid*.

Value Returned

t	Always returns t.

Example

See axlFormCreate on page 552 and axlFormBuildPopup on page 558 for examples.

axlFormAutoResize

```
axlFormAutoResize(
     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

r_form	Form handle.
t_field	Form field name (string).

Value Returned

t	Form resized.
nil	*r_form* 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
- INTSLIDEBAR
- 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 63 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 63) and *GRID* (See Using Grids on page 518) controls.

Important

Please note the following restrictions:

- Setting the same or close text and background colors can cause readability issues.
- Setting the background of `CHECKLIST` controls is not supported on UNIX.
- Dialog boxes with popups do not correctly show color.

Arguments

o_form	Form handle.
t_field	Field name.
g_option	Option (see above)
g_color	Color (see `axlColorDoc`) or `nil`

Value Returned

`t`	Color changed.
`nil`	Error due to an incorrect argument.

Example 1

You can find an example in `axlform.il`.

```
axlFormColorize(f1s "string" 'text 'red)
```

Sets text of string control to red.

Example 2

```
axlFormColorize(f1s "string" 'background 'green)
```

Sets background of string control to green.

Example 3

```
axlFormColorize(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

r_form	Form's *dbid*.
t_field	Form field name (string).

Value Returned

t_field	Active field name.
nil	No active field.

axlFormGridBatch

```
axlFormGridBatch(
     r_cell
)
⇒ t/nil
```

Description

Always used with `axlFormGridSetBatch`. Sets many grid cells efficiently.

Arguments

`r_cell`	Obtained from `axlFormGridNewCell`.

Value Returned

`t`	Grid cells set.
`nil`	No grid cells set.

axlFormGridCancelPopup

```
axlFormGridCancelPopup(
     r_form
     t_field

)

⇒ t/nil
```

Description

After any change to grid content, the application must tell the grid that the changes are complete. The grid then updates itself to the user. Changes include: adding or deleting columns and changing cells.

Arguments

r_form	Form handle.
t_field	Field name.

Value Returned

t	Success.
nil	Failure due to incorrect arguments.

axlFormGridDeleteRows

```
axlFormGridDeleteRows(
     r_form
     t_field
     x_row
     x_number
)
⇒ t/nil
```

Description

Deletes *x_number* rows at *x_row* number. *x_row*=,*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

r_form	Form handle.
t_field	Field name.
x_row	Row number to start delete.
x_number	Number of rows to delete.

Value Returned

t	Rows deleted.
nil	No rows deleted.

axlFormGridEvents

```
axlFormGridEvents(
     r_form
     t_field
     s_event/(s_event1 s_event2 ...)
)
⇒ t/nil
```

Description

Sets user events of interest. It is critical for your application to only set the events that you actually process since enabled events are scripted.

Grid events include the following:

`'rowselect`	Puts grid into row select mode. This is mutually exclusive with `cellselect`.
`'cellselect`	Puts grid into cell select mode. This is mutually exclusive with `rowselect`.
`'change`	Enables cell change events. Use this option if you have check box and text box type cells.
`'rightpopup`	Enables right mouse button popup. A popup must have been specified in the form file.
`'rightpopupPre`	Enables callback to application before a right mouse popup is displayed. This allows the user to modify the popup shown. Also requires `'rightpopup` be set.
`'leftpopupPre`	Enables callback to application before a left mouse popup is displayed. This allows the user to modify the popup shown. Left mouse popups are only present in the drop down cell type.

By default, the grid body has `rowselect` enabled while the headers have nothing enabled.

The form callback structure (*r_form*) has the following new attributes that are only applicable for grid field types:

Event	Row	Col	<Data Fields>
rowselect	<*row*>	1	No
cellselect	<*row*>	<*col*>	Yes (1)
change	<*row*>	<*col*>	Yes (1)
rightpopup	<*row*>	<*col*>	Yes
rightpopupPre	<*row*>	<*col*>	No (2) (3)
leftpopupPre	<*row*>	<*col*>	No (2) (3)

where:

<*row*>	Row number (1 based)
<*col*>	Column number (1 based)
<*Data fields*>	Setting of the *r_form* attributes *curValue*, *curValueInt* and *isValueString*.

- Communicates the value of the data *before* the field is changed. The change event sends the value *after* the field is changed.
- Events are sent immediately before a popup is displayed so the application has the opportunity to modify it. See axlFormGridEvents on page 594 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 518 for a grid overview.

Note: Assigning events to a grid overrides the previous assignment.

This does not work:

```
axlFormGridEvents(fw "grid 'change)
axlFormGridEvents(fw "grid 'cellselect)
```

This works:

```
axlFormGridEvents(fw "grid '(cellselect change) )
```

Arguments

r_form	Form handle.
t_field	Field name.
s_events	See above.

Value Returned

t	User event set.
nil	No user event set.

axlFormGridGetCell

```
axlFormGridGetCell(
     r_form
     t_field
     r_cell
)
⇒ r_cell/nil
```

Description

Returns grid cell data for a given row and column. All associated data for the cell is returned.

Note: The cell value is always returned as a string except for REAL and LONG data types which are returned in their native format.

If row or cell number of 0 is used then top or side heading data is returned (if present.)

Note: For best performance, reuse the cell if accessing multiple cells.

Arguments

r_form	Form handle.
t_field	Field name.
r_cell	Grid cell from `axlFormGridNewCell()`.

Value Returned

r_cell	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 518 for an overview.

Table 10-5 on page 599 describes the `FormGridCol` attributes.

Table 10-5 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 `axlFormGridSet`.
`Alignment Types (left, right, and center):`			
`align`	symbol	Left	Column alignment.
`topAlign`	symbol	Center	Column header alignment.

Table 10-5 FormGridCol Attributes

Attribute	Type	Default	Description
`scriptLabel`	string	<*row number*>	Column scripting name. If the column entry can be edited, you can provide a name which is recorded to the script file. For fieldTypes of TEXT, this option is ignored. Case is ignored and text should not have whitespace or the symbol '`!`'.
`popup`	string	n/a	Name of the associated popup. May be applied to columns or cells of types ENUMSET, STRING, LONG, or REAL.
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

`r_form`	Form handle.
`t_field`	Field name.
`r_formGridCol`	Instance of type `formGridCol`.

Value Returned

`t`	Column added.

`nil`	Failure due to a nonexistent form or field, field not of type `GRID`, errors in the *`g_options`* defstruct, or grid already had a row added.

Examples

For a complete grid programming example, see: `<cdsroot>/share/pcb/examples/skill/ui`.

Example 1

```
options = make_formGridCol
options->fieldType = 'TEXT
options->align = 'center
axlFormGridInsertCol(r_form "grid" options)
```

Adds the first column of type `TEXT` (non-editable) with center alignment.

Example 2

```
options->fieldType = 'ENUMSET
options->popup = "grid2nd"
options->colWidth = 10
options->scriptLabel = "class"
axlFormGridInsertCol (r_form "grid" options)
```

Adds the second column of type `ENUM` (non-editable) with column width of 10 and center alignment, assuming that the form file has a popup definition of *`grid2nd`*.

axlIsGridCellType

```
axlIsGridCellType(
     r_cell
)
⇒ t/nil
```

Description

Tests the passed symbol to see if its user type is of the form `"grid cell"`.

Arguments

r_cell	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

r_form	Form handle.
t_field	Field name.
x_row	Row number of insertion point.
x_number	Quantity of rows to add.

Value Returned

t	One or more rows inserted.
nil	No rows inserted.

axlFormGridNewCell

```
axlFormGridNewCell(
)
⇒ r_cell
```

Description

Creates a new instance of *r_cell* which is required as input to `axlFormGridBatch` or `axlFormSetField` for form grid controls. As a convenience, the consuming APIs do not modify the cell attributes. You need not reset all attributes between API calls.

See axlFormGridSetBatch on page 607 for a complete description of cell attributes.

Arguments

None.

Value Returned

r_cell	New list gridcell handle.

axlFormGridReset

```
axlFormGridReset(
     r_form
     t_field

)

⇒ t/nil
```

Description

Resets grid to its unloaded state. Application should then set the columns, then rows, to the same state as when they initially loaded the windows.

Changes the number of columns after the grid has already been initialized.

Arguments

r_form	Form handle.
t_field	Field name.

Value Returned

t	Grid reset.
nil	Grid not reset.

Example

For a programming example, see `fgrid.il` in `<`*cdsroot*`>/share/pcb/examples/skill/`

Pseudo code:

```
axlFormGridReset(fg "grid")
initCols()
initRows()
axlFormGridUpdate(fg "grid")
```

axlFormGridSetBatch

```
axlFormGridSetBatch(
     r_form
     t_field
     s_callback
     g_pvtData

)

⇒ t/nil
```

Description

Changes grid cells much faster than `axlFormSetField` when changing multiple cells. Both APIs require a grid cell data type (`axlFormGridNewCell`.)

Grid performs single callback using *`s_callback`* to populate the grid. You must call `axlFormGridBatch` in the callback in order to update grid cells.

See the programming example, `grid.il` at <*`cdsroot`*>`/share/pcb/examples/skill/ui`. Create rows and columns before calling this batch API.

Within the callback, use only `axlFormNewCell` and `axlFormGridBatch` from the `axlForm` API.

After changing cells, update the display using `axlFormGridUpdate` outside of the callback.

Grid Cell Data Type (r_cell) Attributes

`x_row`	Row to update.
`x_col`	Column to update.
`g_value`	Value (may be string, integer, or float) if `nil`, preserve current grid setting for the cell.
`s_backColor`	Optional background color.
`s_textColor`	Optional text color.
`s_check`	Set or clear check mark for `CHECKITEM` cells. Ignored for non-check cells. Value may be `t` or `nil`.
`s_noEdit`	If cell is editable, disables edit. Ignored for `TEXT` columns since they are not editable. Current settings are preserved.

`s_invisible`	Make cell invisible. Current cell settings are preserved by the grid.
`s_popup`	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.
`t_objType`	Object name `"r_cell"` (read-only)

Note: Previous grid cell settings are overridden by values in `s_noEdit` and `s_invisible`.

Column and Row access:

Rows and columns are 1 based. To set the cell in the first column and row, you set the row and col number to 1.

You can control header and script text with reserved row and column values as follows:

(<*`row`*>, 0)	Set side header display text.
(<*`row`*>, -1)	Set side header scripting text.

Note: Case is ignored, and text must not contain spaces or the `'!'`

(0, <*`col`*>)	Set top header display text. You may also set the top header at column creation time using `axlFormGridInsertCol`.
(0, 0)	Setting not supported.

For headers and script text, *`g_value`* is the only valid attribute other than *`row`* and *`col`*.

Colors available for `s_backColor` and `s_textColor`:

- nil - use system defaults for color, typically white for background and black for text
- black
- white
- red
- green
- yellow
- button - use the current button background color

Arguments

`r_form`	Form handle.
`t_field`	Field name.
`t_callback`	Function to callback. Takes a single argument: *`g_pvtData`*
`g_pvtData`	Private data (Pass `nil` if not applicable.)

Value Returned

`t`	Grid cell changed.
`nil`	No grid cell changed, or application callback returned `nil`.

axlFormGridUpdate

```
axlFormGridUpdate(
          r_form
          t_field
) -> t/nil
```

Description

Unlike the form lists control you must manually notify the grid control that it must update itself. You should use this call in the following situations:

- Inserting a row or rows
- Deleting a row or rows
- Changing cell(s)

You should make the call at the end of all of changes to the grid.

Do not make this call inside the function you use with axlFormGridSetBatch. Make it after axlFormGridSetBatch returns.

Arguments

r_form	Standard form handle.
t_field	Standard field name.

Value Returned

Returns `t` for success, `nil` for failure.

See Also

`axlFormGridNewCell`

axlFormInvalidateField

```
axlFormInvalidateField(
     r_form
     t_field
)
⇒ t/nil
```

Description

Invalidates the form's field. Allows Windows to send a redraw message to the field's redraw procedure.

Use only for thumbnail fields.

Arguments

r_form	Form handle.
t_field	Field name.

Value Returned

t	Field invalidated.
nil	No field invalidated.

axlFormIsFieldEditable

```
axlFormIsFieldEditable(
     r_form
     t_field
)
⇒ t/nil
```

Description

Checks whether the given form field is editable. If the field is editable, `t` is returned. If the field is greyed, then `nil` is returned.

Arguments

r_form	Form id.
t_field	Field name.

Value Returned

`t`	Field is editable.
`nil`	Field is greyed, or not editable.

axlFormListAddItem

```
axlFormListAddItem(
     r_form
     t_field
     t_listItem/lt_listItems/nil
     g_index
)
⇒ t/nil
```

Description

Adds an item to a list at position *x*. To add many items efficiently, pass the items as a list.

Arguments

r_form	Form id.
t_field	Field name.
t_listItem	String of items in the list. If adding to list for the first time, you must send a `nil` to display the list.
lt_listItems	List of strings to add.
g_index	0 = First item in the list. -1 = Last item in the list.

Value Returned

`t`	One or more items added to list.
`nil`	No items added to list due to incorrect arguments.

Example 1

```
axlFormListAddItem(f1, "list" "a" -1)
axlFormListAddItem(f1, "list" "b" -1)
axlFormListAddItem(f1, "list" "c" -1)
; since first time, send a nil to display the list
axlFormListAddItem(f1, "list" nil, -1)
```

Adds three items to the end of a list.

Example 2

```
axlFormListAddItem(f1, "list" '("a" "b" "c"), -1)
```

Adds three items to the end of a list (alternate method).

axlFormListDeleteItem

```
axlFormListDeleteItem(
     r_form
     t_field
     t_listItem/x_index/lt_listItem/nil
)
⇒ t/x_index/nil
```

Description

Deletes indicated item in the list. You can delete by a string or by position. Deleting by string works best if all items are unique. Position can be problematic if you have the list sort the items that you add to it.

To quickly delete multiple items, call this interface with a list of items.

Note: Delete by list only supports a list of *strings*.

Arguments

r_form	Form id.
t_field	Field name.
x_index	Position of the item to be deleted. 0 is the first item in the list, -1 is the last item in the list.
t_listItem	String of item to delete.
lt_listItems	List of items to delete.
nil	Deletes the last item.

Value Returned

x_index	If using strings (*t_listItem*) to delete items, returns the index of strings deleted. Useful for allowing the code to automatically select the next item in the list.
t	If deleting by index (*x_index*), it returns t if successful in deleting the item.
nil	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

r_form	Form id.
t_field	Field name.
x_index	Offset into the list. 0 = First item in the list. -1 =Last item in the list.

Value Returned

t_listItem	String of item in the list.
nil	Index not valid, or no item at that index.

axlFormListGetSelCount

```
axlFormListGetSelCount(
          r_form
          t_field
          )
==> x_count/nil
```

Description

This only applies to a multi-select list box (OPTIONS multiselect in form file). Returns a count of number of items selected in a multi-select list box.

Arguments

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

Example

See `axlform.il` example.

axlFormListGetSelItems

```
axlFormListGetSelItems(
          r_form
          t_field
          )
==> lt_selected/nil
```

Description

This only applies to a multi-select list box (OPTIONS multiselect in form file).

For a multi-select list box returns list of strings for items selected. If no items selected or this is not appropriate for control returns `nil`.

Arguments

r_form	Form control.
t_field	Name of the field.

Value Returned

lt_selected	List of strings for items selected.
`nil`	Error or nothing selected.

See Also

axlFormListGetSelCount, axlFormListSelAll

Example

See `axlform.il` example.

axlFormListOptions

```
axlFormListOptions(
     r_form
     t_field
     s_option/(s_option1 s_option2 ...)
)
⇒ t/nil
```

Description

Sets options for a list control. The following options are supported:

'*doubleClick*	Enable double-click selection. Passing a `nil` for an option sets default list behavior. Default is single click.

Double-click events are handled as follows:

- Receive the first click as an event with the item selected and the result is: `doubleClick = nil`.
- Receive the second click as an event with the item selected and the result is: `doubleClick = t`.

Suggested use model:

On first click do what would normally happen if the user clicks only once. The second click is a natural extension. For example, on a browser the first click selects the file. The second click does what the *OK* button would do: send the file to the application and close the form.

Arguments

r_form	Form id.
t_field	Field name.
s_option	Sets option for list control. `nil` 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

r_form	Form control.
t_field	Name of the field.
g_set	`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 505.) 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

r_form	Form *dbid*.
t_messageLabel	Message label defined in form file by the MESSAGE keyword.
g_arg1 ...	Substitution parameters (see `printf`)

Value Returned

t_msg	Message that prints.
`nil`	No message with the given name found in this form file.

Examples

```
Form file (level: 0 is info, 1 is info with no journal entry, 2 is warning, 3 is error, and 4 is
fatal.);
     MESSAGE drccount 0 "Drc Count of %d for %s"
     MESSAGE drcerrors 2 "Drc Errors"


axlFormMsg(fw "drccount" 10 "spacing")
axlFormMsg(fw "drcerrors")
```

axlFormGetFieldType

```
axlFormGetFieldType(
     r_form
     t_field
)
⇒g_fieldType/nil
```

Description

Returns the control type for a form field. See the keywords in
Callback Procedure: formCallback on page 581 for a list of supported field types.

Arguments

r_form	Form *dbid*.
t_field	Field name.

Value Returned

g_fieldType	One of the control types.
nil	Field does not exist or is not one of the types supported.

axlFormDefaultButton

```
axlFormDefaultButton(
     r_form
     t_field/g_mode
)
⇒ t/nil
```

Description

Forms normally automatically set a *default button* in a form with the `DEFAULT` section in the form file or with the *OK* and *DONE* labels. When the user hits a carriage return, the *default button* is executed.

A form can have, at most, one default button. Only a field of type `BUTTON` can have the default button attribute.

Note: If default buttons are disabled in a form, then attempts to establish a new default button are ignored. You can only change the default button if the capability in the form is enabled.

Arguments

r_form	Form *dbid*.
t_field	Field name to establish as new button default.
g_mode	`t` to enable the default button in the form, `nil` to disable it.

Value Returned

`t`	Default button set.
`nil`	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 518 for a grid overview.

Arguments

r_form	Form handle.
t_field	Field name.
s_name/*g_value*	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

r_form	Form *dbid*.
t_field	Field name.

Value Returned

`t`	Field set active.
`nil`	Failed to set the field active.

axlFormSetDecimal

```
axlFormSetDecimal(
     o_form
     g_field
     x_decimalPlaces
)
⇒ t/nil
```

Description

Sets the decimal precision for real fill-in fields in the form. If *g_field* is `nil`, sets the precision for all real fill-in fields in the form.

Arguments

o_form	Form handle.
g_field	Field label, or `nil` for all fields.
x_decimalPlaces	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 greyed (`nil`).

Arguments

r_form	Form *dbid*.
t_field	Field name.
g_editable	Editable (`t`) or greyed (`nil`).

Value Returned

`t`	Set form field to editable or greyed.
`nil`	Failed to set form field to editable or greyed.

axlFormSetFieldLimits

```
axlFormSetFieldLimits(
     o_form
     t_field
     g_min
     g_max

)

⇒ t/nil
```

Description

Sets the minimum or maximum values a user can enter in an integer or real fill-in field. If a `nil` value is provided, that limit is left unchanged.

For a `REAL` field, the type for *`g_min`* and *`g_max`* may be `int`, `float`, or `nil`. For an `INT` or `LONG`, the type must be `int` or `nil`.

Arguments

`o_form`	Form handle.
`t_field`	Field label.
`g_min`	Minimum value for field.
`g_max`	Maximum value for field.

Value Returned

`t`	Set max or min.
`nil`	Error due to invalid argument(s).

axlFormTreeViewAddItem

```
axlFormTreeViewAddItem(
     r_form
     t_field
     t_label
     g_hParent
     g_hInsertAfter
     [g_multiSelectF]
     [g_hLeafImage]
     [g_hOpenImage]
     [g_hClosedImage]
)
⇒ g_hItem/nil
```

Description

Adds an item to a treeview under *parent* and after *insertAfter* sibling. If sibling is `nil`, item is added as the last child of a parent. If parent is `nil`, item is created as the root of the tree.

Note: You must use this to add an item to a tree. `axlFormSetField` is disabled for Tree controls.

Applications must keep the returned handle *`l_hItem`* since a handle will be passed as *`form->curValueInt`* when the item is selected from tree view. The string associated with the selected item is also passed as *`form->curValue`*, however the string value may not be unique and cannot be used as a reliable identifier for the selected treeview item.

Note: Tree view defaults to single selection mode. There is no checkbox associated with items in the tree view to make multiple selections. To make a tree view item multi select, you pass one of the following values for *`t_multiSelectF`*:

- `nil` or `'TVSELECT_SINGLE` for no selection state checkbox
- `t` or 'TVSELECT_2STATE for 2 state checkbox
- `'TVSELECT_3STATE` for 3 state checkbox

If an item is defined as multi select, a check box appears as part of the item. The user can check/uncheck (2 state) this box to indicate selection or select checked/unchecked/disabled modes for a 3 state checkbox. When the user makes any selection in the checkbox, its value is passed to application code in *`form->treeViewSelState`*. In this case, *`form->curValue`* is `nil`.

Arguments

r_form	Form *dbid*.
t_field	Field name.
t_label	String of item in the treeview.
g_hParent	Handle of parent. If `null`, item created as the root of the tree.
g_hInsertAfter	Handle of the sibling to add the item after. If `null`, item added at the end of siblings of the parent.
t_multiSelectF	If `t`, the item has a checkbox for multi selection.
g_hLeafImage	Handle of the image to use whenever a leaf node in the tree view. If `nil` or not supplied, the default pink diamond image is used.
g_hOpenImage	Handle of image to use whenever an expanded parent node in the tree view. If `nil` or not supplied, the default open folder image is used.
g_hClosedImage	Handle of image to use whenever an unexpanded parent node in the tree view. If `nil` or not supplied, the default closed folder image is used.

Value Returned

g_hItem	Item is added to the tree view control.
`nil`	No item is added to the tree view control due to an error.

Examples

Creates a treeview.

Constructs a list that resembles a tree structure. Each list element is composed of the item label and the handle that was returned from a call to `axlFormTreeViewAddItem()`. The structure is for the following tree sample:

```
"For Sale"
     "Redmond"
          "100 Berry Lane"
          "523 Apple Road"
          "1212 Peach Street"
     "Bellevue"
          "22 Daffodil Lane"
          "33542 Orchid Road"
          "64134 Lily"
     "Seattle"
          "33 Nicholas Lane"
          "555 Tracy Road"
          "446 Jean Street"


(defstruct myTreeItem string id)

(defun myTreeItems ()
(list
(make_myTreeItem ?string "For Sale" ?id nil)
(list (make_myTreeItem ?string "Redmond" ?id nil)
     (list (make_myTreeItem ?string "100 Berry Lane" ?id nil))
     (list (make_myTreeItem ?string "523 Apple Road" ?id nil))
     (list (make_myTreeItem ?string "12 Peach Street" ?id nil)))
(list (make_myTreeItem ?string "Bellevue" ?id nil)
     (list (make_myTreeItem ?string "22 Daffodil Lane" ?id nil))
     (list (make_myTreeItem ?string "354 Orchid Road" ?id nil))
     (list (make_myTreeItem ?string "64 Lily Street" ?id nil)))
(list (make_myTreeItem ?string "Seattle" ?id nil)
     (list (make_myTreeItem ?string "33 Nicholas Lane" ?id nil))
     (list (make_myTreeItem ?string "5 Tracy Road" ?id nil))
     (list (make_myTreeItem ?string "46 Jean Street" ?id nil)))))

(defun axlAddTreeItem (form field parent item)
(let (hParent hItem hAfter str)
     (cond
     ((get item 'string)
          (setq str (get item 'string))
          (setq hParent (get parent 'id))
          (setq hAfter nil) ; for now
          (setq hItem (axlFormTreeViewAddItem form field str hParent hAfter))
          item->id = hItem
          (t
```

```
          nil))))

(defun axlAddTreeItems (form field l_list)
(let ((parent (car l_list)))

     ;; root node
     (axlAddTreeItem form field nil parent)

     ;; Add the reset of the nodes
     (axlAddTreeItemsRecursive form field l_list)))

(defun axlAddTreeItemsRecursive (form field l_list)
(let ((parent (car l_list)))
     (when parent
          (mapc
               (lambda (item)
                    (axlAddTreeItem form field parent (car item))
                    (axlAddTreeItemsRecursive form field item))
               (cdr l_list)))))
```

Callback Values

In your callback function, the form has the following properties relevant to treeviews:

form->curValue	Contains label of a treeview item. Set in single select mode and in multi select mode when the user selects the item. In this case, the *result->tree.selectState* is -1.
form->curIntValue	Contains id of the selected treeview item.
form->selectState	In multi select mode, when the user picks the selection checkbox, this field contains: 0 if selection checkbox is not checked 1 if selection checkbox is checked 2 if selection checkbox is disabled In this case the *result->string* is empty. In all other cases the value is -1.
form->event	If event property is set to `"rightpopup"`, treeview control has a popup and user has selected an item in the popup. In this case, *form->curValue* is set to the popup index selected, and *form->selectState* is set to `-1`. *form->curIntValue* is set to the treeview item id.

You add popups to treeview fields in a form like you add any other field in a form.

For all non-popup operations, event is set to `"normal."`

axlFormTreeViewChangeImages

```
axlFormTreeViewChangeImages(
     r_form
     t_field
     g_hItem
     [g_hLeafImage]
     [g_hOpenImage]
     [g_hClosedImage]
)
⇒ t/nil
```

Description

Modifies various bitmap images associated with a given tree view item.

Arguments

r_form	Form id.
t_field	Field name.
g_hItem	Handle of item in tree view. Handle was returned as a result of the call to `axlFormTreeViewAddItem` when item was initially added.
g_hLeafImage	Handle of the image to use whenever item is a leaf node in the tree view. If `nil` or not supplied, the default pink diamond image is used.
g_hOpenImage	Handle of image to use whenever item is an expanded parent node in the tree view. If `nil` or not supplied, the default open folder image is used.
g_hClosedImage	Handle of image to use whenever item is an unexpanded parent node in the tree view. If `nil` 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.

axlFormTreeViewChangeLabel

```
axlFormTreeViewChangeLabel(
     r_form
     t_field
     g_hItem
     t_label

)

⇒ t/nil
```

Description

Modifies text of a given treeview item.

Arguments

r_form	Form id.
t_field	Field name.
g_hItem	Handle of item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem`.
t_label	New label.

Value Returned

`t`	Tree view item's label is modified.
`nil`	Failed to modify tree view item's label.

axlFormTreeViewGetImages

```
axlFormTreeViewGetImages(
     r_form
     t_field
     g_hItem
)
⇒ l_hImage/nil
```

Description

Returns various bitmap image handles that refer to images used by a specified item in the tree view.

Arguments

r_form	Form id.
t_field	Field name.
g_hItem	Handle of item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem` when this item was initially added.

Value Returned

l_hImage	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.
`nil`	Error due to invalid arguments.

axlFormTreeViewGetLabel

```
axlFormTreeViewGetLabel(
     r_form
     t_field
     g_hItem
)
⇒ t_label/nil
```

Description

Returns text of a given treeview item.

Arguments

r_form	Form id.
t_field	Field name.
g_hItem	Handle of item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem` when this item was initially added.

Value Returned

t_label	Text of given tree view item.
`nil`	Failed to get text of given tree view item due to invalid arguments.

axlFormTreeViewGetParents

```
axlFormTreeViewGetParents(
     r_form
     t_field
     g_hItem
)
⇒ lg_hItem/nil
```

Description

Returns a list of all the ancestors of a treeview control item, starting from the root of the tree. Helps in search operations in SKILL. Applications can traverse their tree list following parent lists to a given item instead of searching the whole tree for an item.

Arguments

r_form	Form id.
t_field	Field name.
g_hItem	Handle of item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem` when this item was initially added.

Value Returned

lg_hItem	List containing all parents, starting from the root.
`nil`	Failed to obtain list due to invalid arguments.

axlFormTreeViewGetSelectState

```
axlFormTreeViewGetSelectState(
     r_form
     t_field
     g_hItem
)
⇒ x_selectState
```

Description

In multi select mode, returns the select state. This is different than the current selected item in single select tree views. In multi select mode, users can change the select state by clicking on the select checkbox associated with each item.

Arguments

r_form	Form id.
t_field	Field name.
g_hItem	Handle of item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem` when this item was initially added.

Value Returned

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

r_form	Form id.
t_field	Field name.
lt_bitmaps	Either a string containing the name of the bitmap file to load, or a list of strings, each of which is the name of a bitmap file to load.

Notes:

- Bitmap images must be 16 x 16 pixels.
- A bitmap file may contain more than one image provided they are appended horizontally (for example, a bitmap file containing *n* images will be (16**n*) x 16 pixels).
- Color RGB (255,0,0) is reserved for the transparent color. Any pixel with this color is displayed using the background color.

Value Returned

l_hImage	List of image handles that the caller will use to reference the images in subsequent `axlFormTreeViewAddItem` calls. List is ordered to correspond with the order that the images were listed in the *lt_bitmaps* parameter.
`nil`	One or more of the bitmap files could not be found, or an error was encountered while adding images.

Example

Given the file `myBmp1.bmp` is a 16 x 16 bitmap and `myBmp2.bmp` is a 32 x 16 bitmap.

```
(axlFormTreeViewLoadBitmaps (list "myBmp1" "myBmp2"))
```

This might return the following:

```
(6 7 8)
```

You can interpret the returned handles as follows:

- Image handle 6 refers to `myBmp1.bmp`.
- Image handle 7 refers to the left half of `myBmp2.bmp`.
- Image handle 8 refers to the right half of `myBmp2.bmp`.

axlFormTreeViewSet

```
axlFormTreeViewSet(
     r_form
     t_field
     s_option
     g_hItem
     [g_data]
)
⇒ t/nil
```

Description

Allows an application to change global and individual items in a tree view control.

Arguments

r_form	Form id.
t_field	Field name.
s_opton and *g_hItem*	*s_option* is one of the symbols listed. Each is paired with *g_hItem*, the handle of an item in the tree view control. If the option you are setting is global, you use `nil` in place of *g_hItem*.

- `TV_REMOVEALL` (`nil`)
- `TV_DELETEITEM` (*g_hItem*)
- `TV_ENSUREVISIBLE` (*g_hItem*)
- `TV_EXPAND` (*g_hItem*)
- `TV_COLLAPSE` (*g_hItem*)
- `TV_SELECTITEM` (*g_hItem*, or pass `nil` to deselect item)
- `TV_SORTCHILDREN` (*g_hItem*)
- `TV_NOEDITLABEL` (`nil` - disables in place label editing)
- `TV_NOSELSTATEDISPATCH` (`nil` -check box selection not dispatched)
- `TV_ENABLEEDITLABEL` (`nil` - enable in place label editing)

- TV_MULTISELTYPE (*g_hItem*)
- TV_NOSHOWSELALWAYS (nil)
- TV_SHOWSELALWAYS (nil)

Note: For TV_MULTISELTYPE, you can also use the option *g_data* which is one of the following: TVSELECT_SINGLE, TVSELECT_2STATE, TVSELECT_3STATE. Default is TVSELECT_SINGLE.

g_hItem	Handle of item in the tree view control.

Note: You get *g_hItem* in the following ways:

- Call axlFormTreeViewAddItem.
- Call axlFormTreeViewGetParents.
- Change a tree control that causes a form dispatch. Then the form User Type attributes curValue and curValueInt are the *g_hItem*.
- You can pass nil for *g_hItem* in some cases. Pass nil for TV_SELECTITEM option to deselect the item that is currently selected.

Value Returned

t	Changed one or more items in tree view control.
nil	Failed to change items in tree view control.

Example

```
(axlFormTreeViewSet form form->curField 'TV_DELETEITEM form->curValue)
```

Deletes selected treeview item in a form.

For more examples see <*cdsroot*>/share/pcb/examples/skill/form/basic

axlFormTreeViewSetSelectState

```
axlFormTreeViewSetSelectState(
     r_form
     t_field
     g_hItem
     g_state
)
⇒ t/nil
```

Description

In multi select mode, sets the select state. This is different than the current selected item in single select tree views. In multi select mode, users can change the select state by clicking on the select checkbox associated with each item.

Arguments

r_form	Form id.
t_field	The name of the field.
g_hItem	Handle of the item in the tree view. This handle was returned as a result of the call to `axlFormTreeViewAddItem` when this item was initially added.
g_state	Select state to set.

- Select state is unchecked if *g_state* is `nil` or `'TVSTATE_UNCHECKED`
- Select state is checked if *g_state* is `t` or `'TVSTATE_CHECKED`
- Select state is disabled if *g_state* is `'TVSTATE_DISABLED`

Value Returned

`t`	Set select state.
`nil`	Failed to set select state.

11

Simple Graphics Drawing Functions

Overview

This chapter describes the AXL-SKILL functions related to Simple Graphics Drawing. You use these drawing utilities for drawing into bitmap areas such as thumbnails within the UIF forms package.

`axlGRP` is the AXL interface to a Simple Graphics Drawing utility.

You can simplify application drawing into thumbnails within forms as follows:

1. Specify the thumbnail field within the form file. This should not have a bitmap associated with it.
2. Call the `axlGRPDrwInit` function with the form, field name, and a callback function. Keep the handle returned by this function so you can use it in later processing.
3. Using the functions provided, redraw the image. The callback function is invoked with the graphics handle as the parameter.
4. Use the `axlGRPDrwUpdate` function to trigger the callback function.

Note: The application *cannot* set bitmaps or graphics callbacks using the facilities outlined in the Thumbnail documentation while using this package.

Simple Graphics Drawing package supports the following:

- Rectangles (Filled and unfilled)
- Polygons (Filled and unfilled)
- Circles (Filled and unfilled)
- Simple Lines
- Poly Lines
- Bitmaps
- Text

This package supports a mappable coordinate system. With the `GRPDrwMapWindow` function, you can specify a rectangle that gets mapped to the actual drawing area. An aspect ratio of 1 to 1 is maintained.

The zero point of the drawing area is the upper left point of the drawing:

Note: Objects drawn earlier are overlapped by objects drawn later. The application must manage this.

Setting Option Properties on the r_graphics Handle

You can set option properties on the *r_graphics* handle before calling the drawing functions. These properties, if applicable, are used by the drawing properties.

Table 11-1 r_graphics Option Properties

Option	Default	Description	Available Settings
color	black	Applies to all elements.	black, white, red, green, yellow, blue, lightblue, rose, purple, teal, darkpink, darkmagenta, aqua, gray, olive, orange, pink, beige, navy, violet, silver, rust, lime, brown, mauve, magenta, lightpink, cyan, salmon, peach, darkgray, button (represents the current button color)
fill	unfilled	Applies only to rectangles and polygons.	filled, filled_solid, unfilled
width	0	Applies only when geometric elements are unfilled.	
text_align	left	Text justification. Applies only to text.	left, center, right
text_bkmode	transparent	Text background display mode. Applies only to text.	transparent, opaque

Examples

See `<cdsroot>/share/pcb/examples/skill/form/basic/axlform.il` for a complete example.

Functions

axlGRPDrwBitmap

```
axlGRPDrwBitmap(
     r_graphics
     t_bitmap
)
⇒ t/nil
```

Description

Loads a bitmap into the drawing area in the graphics field. More drawing can take place on top of the bitmap.

Arguments

r_graphics	Graphics handle.
t_bitmap	Name of bitmap file. File must be on the `BMPPATH`, with `.bmp` as the extension.

Value Returned

`t`	Bitmap loaded into drawing area in the graphics field.
`nil`	No bitmap loaded into the drawing area in the graphics field due to invalid arguments.

axlGRPDrwCircle

```
axlGRPDrwCircle(
     r_graphics
     l_origin
     x_radius
)
⇒ t/nil
```

Description

Draws a circle into the area identified by the *r_graphics* handle, at the origin specified, and with the specified radius. Option properties attached to the *r_graphics* handle are applied when drawing the circle.

Arguments

r_graphics	Graphics handle.
l_origin	List noting the *x* and *y* coordinates of the origin.
x_radius	Integer noting the radius of the circle.

Value Returned

t	Circle drawn.
nil	No circle drawn due to invalid arguments.

axlGRPDrwInit

```
axlGRPDrwInit(
     r_form
     t_field
     t_func
)
⇒ r_graphics/nil
```

Description

Sets up necessary data structures for triggering the graphics callback into the graphics field.

Arguments

r_form	Handle of the form.
t_field	Name of field into which the package should draw. (Only THUMBNAIL fields are supported.)
t_func	Name of the drawing callback function. Callback function is invoked with the graphics handle as the parameter.

Value Returned

r_graphics	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

r_graphics	Graphics handle.
l_vertices	List of coordinates describing the line.

Value Returned

t	Line drawn.
nil	No line drawn due to invalid arguments.

axlGRPDrwMapWindow

```
axlGRPDrwMapWindow(
     r_graphics
     x_hgt
     x_width
)
⇒ t/nil
```

Description

Allows the application to denote the coordinate system that is mapped into the drawing area of the graphics field.

Arguments

r_graphics	Graphics handle.
x_hgt	Height of drawing window.
x_width	Width of drawing window.

Value Returned

t	Successful
nil	Error occurred due to invalid arguments.

axlGRPDrwPoly

```
axlGRPDrwPoly(
     r_graphics
     l_vertices

)

⇒ t/nil
```

Description

Draws a polygon into the area identified by the *r_graphics* handle and the list of coordinates. Option properties attached to the *r_graphics* handle are applied when drawing the polygon. If the coordinates do not form a closed polygon, the first and last coordinates in the list are connected by a straight line.

Arguments

r_graphics	Graphics handle.
l_vertices	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

r_graphics	Graphics handle.
l_upper_left	List noting the coordinate of the upper left point of the rectangle.
l_lower_right	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

r_graphics	Graphics handle.
l_origin	List noting the *x* and *y* coordinate of the origin.
t_text	Text string to be drawn.

Value Returned

t	Text drawn.
nil	No text drawn due to incorrect arguments.

axlGRPDrwUpdate

```
axlGRPDrwUpdate(
     r_graphics
)
⇒ t/nil
```

Description

Triggers calling of the application supplied callback function.

Arguments

r_graphics	Graphics handle.

Value Returned

t	Application supplied callback function called.
nil	No callback function called due to an incorrect argument.

12

Message Handler Functions

Overview

This chapter describes the AXL-SKILL message handler system and functions. The message handler system allows you to write AXL-SKILL code that does not have to deal explicitly with errors after each call to a lower level routine, but rather checks at only one or two points. This contrasts with application programs that do not have a buffering and exception-handling message facility, where you must test for and respond to errors and exceptions at each point of possible occurrence in your code. Using the functions described in this chapter, you can do the following:

- Establish a *context* for your application messages.

  A context is a logical section of your application during which you want to buffer and test for the potential errors and warnings you provided for in the lower levels of your application code.

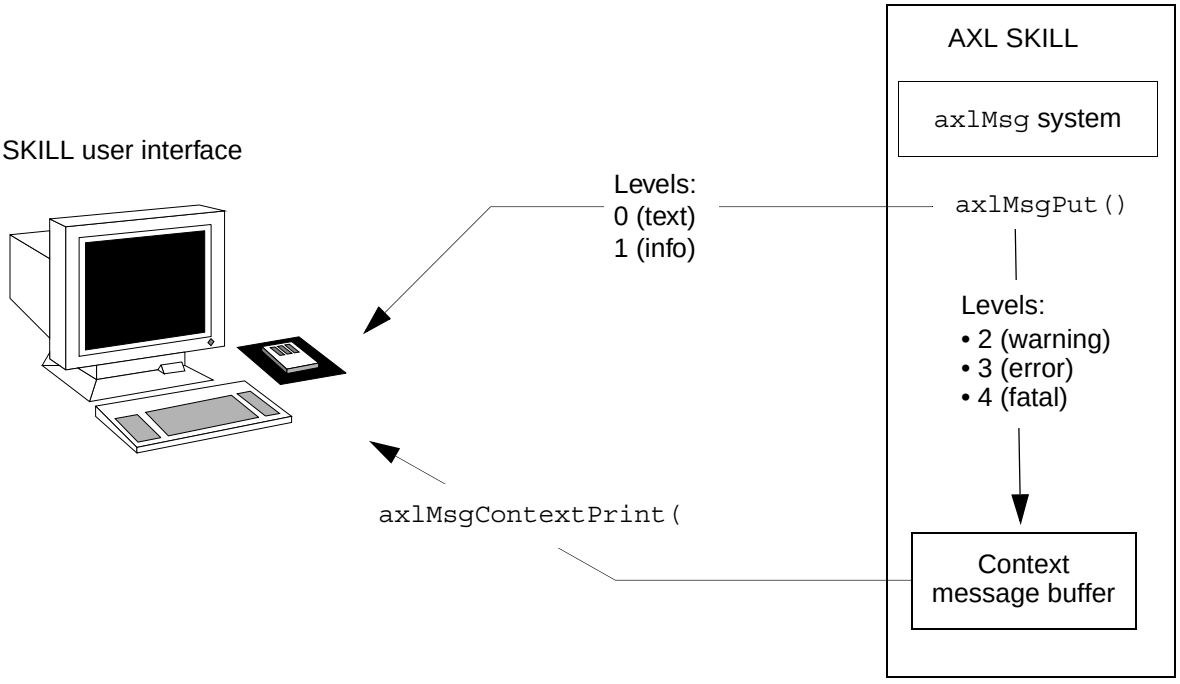
- Write messages to the user with one of five levels of *severity*:

Severity Level	Type	Type Description and Response by AXL Message Handler
0	Text	Data created by the application, such as a log or report. Writes to journal and the user interface without being buffered.
1	Info	Such as “10% done,” “20% done”... Writes to user interface only, *not* 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.

Severity Level	Type	Type Description and Response by AXL Message Handler
3	Error	Such as “Cannot find symbol DIP24_200 in library.” Beeps and writes to context message buffer and journal with an “E” error tag.
4	Fatal	Such as “Disk read error. Cannot continue.” Double beeps and writes to context message buffer and journal with an “F” fatal tag.

- Test and change the severity level of the messages created and buffered in a context.
- Check for specific messages in the message buffer, and respond appropriately to anticipated conditions detected by lower-level functions.
- Close a context.

The following illustration shows how `axlMsg` functions move messages to and from a context message buffer and the SKILL user interface.



Message Handler Example

```
context = axlMsgContextStart("My own context.")
axlMsgPut(list("My warning" 2))
axlMsgPut(list("My error" 3))
printf("Message severity %d",axlMsgContextTest(context))
axlMsgPut(list("My fatal error %s" 4) "BAD ERROR")
if( axlMsgContextInBuf(context "My error")
     printf("%s\n" "my error is there"))
printf("Message severity %d",axlMsgContextTest(context))
axlMsgContextPrint(context)
axlMsgContextFinish(context)
 ⇒ t
```

1. Starts a context with `axlMsgContextStart`
2. Puts a warning, an error, and a fatal error message using `axlMsgPut`
3. Checks for the error message with `axlMsgContextInBuf`
4. Tests for the context severity level with `axlMsgContextTest`
5. Prints the context buffer with `axlMsgContextPrint`
6. Ends with `axlMsgContextFinish`

When you load the SKILL program shown above, the SKILL command line outputs the following:

```
W- My warning
E- My error
F- My fatal error BAD ERROR
Message severity 3
my error is there
Message severity 4
t
```

General usage of the axlMsg System

- Messages first go to the context buffer
- `axlMsgContextPrint` prints them to the SKILL command line
- The contents of the output buffer from any `print` and `printf` data write to the command line when control returns to the command line. That is why the messages "`Message severity 3`," "`my error is there`" and "`Message severity 4`" follow the buffered messages ("`W- My warning`" ...)

Message Handler Functions

This section lists message handler functions.

axlMsgPut

```
axlMsgPut(
     g_message_format
     [g_arg1 ...]
)
⇒t
```

Description

Puts a message in the journal file. Use this function to “print” messages. It buffers any *errors* or *warnings*, but processes other message classes immediately.

Arguments

g_message_format	Context message (`printf` like) format string. See “Overview” on page 665 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 667.

```
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

r_context	Context handle from `axlMsgContextStart`. Prints messages only for this context (or any children). An argument of `nil` causes `axlMsgContextPrint` to look through all contexts.

Value Returned

`t`	Always returns `t`.

Example

See the "Message Handler Example" on page 667 in the beginning of this chapter.

```
axlMsgContextPrint(context)
 ⇒ t
```

Prints the buffered messages in that context to SKILL command line:

```
W- My warning
E- My error
F- My fatal error BAD ERROR
```

axlMsgContextGetString

```
axlMsgContextGetString(
     r_context
)
⇒ lt_messages/nil
```

Description

Gets the messages in the message buffer and removes them from the buffer. Call `axlMsgContextGetString` subsequently to communicate those messages to the user (for example, in a log file).

Arguments

r_context	Context handle from `axlMsgContextStart`. Gets messages only for this context (or any children). An argument of `nil` causes `axlMsgContextGetString` to look through all contexts.

Value Returned

lt_messages	List of the text strings of the buffered messages.
`nil`	No buffered messages found.

Example

See the "Message Handler Example" on page 667.

```
axlMsgContextGetString(context)
⇒ ("My warning" "My error" "My fatal error BAD ERROR")
```

axlMsgContextGet

```
axlMsgContextGet(
     r_context
)
⇒ lt_format_strings/nil
```

Description

Gets the format strings of the buffered messages. (*Not* the messages themselves. Compare the example here and the one shown for `axlMsgContextGetString`.*)* Does not remove the messages from the message buffer, rather it simply provides the caller an alternative to making a number of `axlMsgContextInBuf` calls.

Arguments

r_context	Context handle from `axlMsgContextStart`. Gets messages only for this context (or any children). An argument of `nil` causes `axlMsgContextGet` to look through all contexts.

Value Returned

lt_format_strings	List of format strings of the buffered messages.
`nil`	No buffered messages found.

Example

See the "Message Handler Example" on page 667.

```
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 673 to check for a particular message class.

Arguments

r_context	Context handle from `axlMsgContextStart`. Looks only for messages for this context. If *r_context* is `nil`, `axlMsgContextTest` looks through all contexts.

Value Returned

x_class	The most severe message class of the messages in the message buffer of the given context.

Example

See the Message Handler Example on page 667.

```
printf("Message severity %d\n" axlMsgContextTest(context))
⇒Message severity 4
```

axlMsgContextInBuf

```
axlMsgContextInBuf(
     r_context
     t_format_string
)
⇒ t
```

Description

Checks whether message *t_format_string* is in the message buffer of context *r_context*. Gives the application the ability to control code flow based on a particular message reported by a called function. The check is based on the original format string, not the fully substituted message.

Arguments

r_context	Context handle from `axlMsgContextStart`. Looks only for messages in this context (or any children). If *r_context* is `nil`, `axlMsgContextInBuf` looks through all contexts.
t_format_string	Format string of the message.

Value Returned

`t`	Specified message is in the buffer.
`nil`	Specified message is not in the buffer.

Example

See the “Message Handler Example” on page 667.

```
if( axlMsgContextInBuf(context "My error")
     printf(%s\n" "My error is there"))
```

axlMsgContextRemove

```
axlMsgContextRemove(
     r_context
     t_format_string
)
⇒t
```

Description

Removes a message (or messages) from the buffered messages. Lets you remove messages (usually warnings) from the buffer that you decide, later in a procedure, that you do not want the user to see.

Arguments

r_context	Context handle from `axlMsgContextStart`. Removes messages for only this context (or any children). If *r_context* is `nil`, `axlMsgContextRemove` looks through all contexts.
t_format_string	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

`t`	Always returns `t`.

Example

See the “Message Handler Example” on page 667.

```
axlMsgContextRemove(context, "My fatal error %s")
⇒t
```

axlMsgContextStart

```
axlMsgContextStart(
     g_format_string
     [g_arg1 ...]
)
⇒ r_context
```

Description

Indicates the start of a message context. Prints any buffered messages for the context.

Arguments

g_format_string	Context message (`printf` like) message format string.
g_arg1 ...	Values for the substitution arguments for the format string. Use `axlMsgClear` (and `Test`/`Set`) functions to control code flow if the function returns insufficient values.

Value Returned

r_context	Context handle to use in subsequent `axlMsgContext` calls.

Example

See the "Message Handler Example" on page 667.

```
context = axlMsgContextStart("Messages for %s" "add line")
Messages for add line
5
```

axlMsgContextFinish

```
axlMsgContextFinish(
     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 `axlMsgContextStart`.

Value Returned

`t`	Always returns `t`.

Example

See the "Message Handler Example" on page 667.

```
context = axlMsgContextStart()
... do some things ...
axlMsgContextFinish(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

r_context	Context handle from `axlMsgContextStart`. If *r_context* is `nil`, clears all messages in all contexts.

Value Returned

`t`	Always returns `t`.

Example

See the "Message Handler Example" on page 667.

```
context = axlMsgContextStart()
... do some things ...
axlMsgContextClear(context)
```

axlMsgCancelPrint

```
axlMsgCancelPrint(
)
⇒ t
```

Description

Prints a message informing the user that he requested *cancel*. Since the severity of this message is *Error* (3), AXL writes it to the context buffer as it does other warning, error, and fatal messages. Call this function only from a routine that requests user input.

Arguments

None

Value Returned

`t`	Always returns `t`.

Example

See the "Message Handler Example" on page 667.

```
axlMsgCancelPrint()
```

Sets severity level to 2 (warning) and puts the following into the message buffer (if `axlMsgSys` messages are in English):

```
User CANCEL received
```

axlMsgCancelSeen

```
axlMsgCancelSeen(

)

⇒ t/nil
```

Description

Checks to see if the `axlMsgCancelPrint` message was printed. Does not poll the input stream for a *CANCEL* key. Checks the buffer of current messages for the occurrence of the `axlSsgSys.cancelRequest` message.

Arguments

None

Value Returned

`t`	Requested *cancel* has been seen.
`nil`	No requested *cancel* has been seen.

Example

See the "Message Handler Example" on page 667.

```
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 667.

```
axlMsgClear()
```

axlMsgSet

```
axlMsgSet(
     x_class
)
⇒ t
```

Description

Sets the current error severity level. Use only to reset the severity cleared using `axlMsgClear`.

Arguments

x_class	Severity level.

Value Returned

`t`	Always returns `t`.

Example

See the "Message Handler Example" on page 667.

```
level = axlMsgTest()
; Do something
...
axlMsgSet(level)
```

axlMsgTest

```
axlMsgTest(

)

⇒ 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 `axlMsgClear` or `axlMsgContextClear`/`Print`/`GetMsg` call.

Arguments

None.

Value Returned

x_class	Highest severity level of all the messages currently in the message buffer.

Example

See the "Message Handler Example" on page 667.

```
level = axlMsgTest()
⇒ 4
```

13

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

axlOpenDesign

Example

```
axlCurrentDesign()
⇒ "mem32meg"
```

Gets the name of the currently active layout.

axlDesignType

```
axlDesignType(
     g_detailed
)
⇒ t_type
```

Description

Returns a string giving the type of the active design. The level of the type returned depends on the argument *g_detailed*, as described below.

Arguments

g_detailed	If *g_detailed* is `nil`, returns the high-level types `"LAYOUT"` to denote a layout design, and `"SYMBOL"` to denote a symbol definition design. If *g_detailed* is `t`, returns a detailed value describing the design as follows: For a `layout` type design: `"BOARD"`, `"MCM"`, `"MDD"`(module), or `"TIL"` (tile). For a `symbol` type design: `"PACKAGE"`, `"MECHANICAL"`, `"FORMAT"`, `"SHAPE"`, or `"FLASH"`.

Value Returned

t_type	One of the string values described.

Example 1

```
axlDesignType(nil)
⇒ "LAYOUT"
```

Gets the high-level design type of the current active layout.

Example 2

```
axlDesignType(t)
⇒ "BOARD"
```

Gets the detailed design type.

axlCompileSymbol

```
axlCompileSymbol(
     ?symbol t_name
     ?type t_type

  )
⇒ t_symbolName/nil
```

Description

Compiles and edit checks the current (symbol) design and saves the compiled version on disk under the name *t_name*. If *t_name* is not provided, then *t_name* is the current drawing name. If *t_name* is provided as `nil`, you are prompted for the name. Uses the symbol type in determining some of the edit checks.

Arguments

t_name	Name of the compiled symbol.
t_type	Type of symbol, the default is the current symbol type (see axlDesignType).

Value Returned

t_symbolName	Name of the symbol.
`nil`	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.

axlSaveEnable

```
axlSaveEnable(
          [g_saveEnable]
) -> t/nil
```

Description

This queries or sets the design save design. When the File – Save design menu item is enabled, nil is disabled. It is not recommended that you unset the save enable.

Arguments

g_saveEnable:	Enables save menu (reset of next database save). If nil, disables the save menu. t

If no arguments are provided, then returns current status.

Value Returned

Returns current save setting if query and previous setting if changing the value.

See Also

axlSaveDesign

Example 1

```
query state of save
     state = axlSaveEnable()
```

Example 2

```
Set state of save to t (enable save menu)
     oldState = axlSaveEnable(t)
```

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).

Arguments

t_symbolType	"package", "mechanical", "format", "shape" or "flash"

Value Returned

t_symbolType	Symbol exists.
nil	Error due to incorrect argument.

Note: This function is only available in the symbol editor. This is a SKILL interface into the change symbol type in the Drawing Parameters dialog box in *allegro_symbol*.

axlDBControl

```
axlDBControl(
     s_name
     [g_value]
  )
g_currentValue/ls_names
```

Description

Inquires and/or sets the value of a special database control. If the setting is a value, the return is the old value of the control.

A side effect of most of these controls is that, if an active dialog box displays the current setting, the setting may not be updated. Additional side effects of individual controls are listed. Items will be added in the future. Items currently supported for *s_name* include the following:

Name: *drcEnable*
Value: `t` or `nil`
Set?: Yes
Description: Enables or Disables DRC/CBD system.
Equivalent: Same as status dialog box drc on/off buttons.
Side Effects: Does not perform batch DRC.

Name: *ratnestDistance*
Value: `t` or `nil`
Set?: Yes
Description: Accesses the ratsnest display mode.

`t` - pin to pin

`nil` - closest dangling net cline/via

Equivalent: Same as status dialog box Ratsnest Dist control.
Side Effects: Will recalculate the display when this is changed.

Name: *activeTextBlock*
Value: 1 to *maxTextBlock*
Set?: Yes
Description: Accesses the active text block number.
Equivalent: Same as status dialog box text block.
Side Effects: None

Name: *maxTextBlock*
Value: 16 to 64
Set?: No
Description: Reports the maximum text block number.
Equivalent: None
Side Effects: None

Name: *activeLayer*
Value: <*class*> / <*subclass*>
Set?: Yes
Description: Accesses the current class/subclass.
Equivalent: Same as ministatus display.
Side Effects: None

Name: *defaultSymbolHeight*
Value: float
Set?: Yes
Description: Sets the default symbol height in the database.
Equivalent: Status dialog box's DRC Symbol Height.
Side Effects: Will set DRC out of date, and does not update status dialog box if it is present.

Name: *highlightColor*
Value: 1 to 24
Set?: Yes
Description: Queries/changes the permanent highlight color. Can be used with `axlHighlightObject`
Equivalent: Color form, Display Group, Permanent highlight
Side Effects: Will also change the perm highlight color in the hilite command. Call `axlVisibleUpdate` to update display.

Name: *tempColor*
Value: 1 to 24
Set?: Yes
Description: Queries/changes the temporary highlight color. Can be used with `axlHighlightObject`
Equivalent: Color form, Display Group, Temporary highlight
Side Effects: Call `axlVisibleUpdate` to update display.

Name: *ratsnestColor*
Value: 1 to 24
Set?: Yes
Description: Queries/changes the ratsnest color
Equivalent: Color form, Display Group, Ratsnest Color
Side Effects: Call axlVisibleUpdate to update display.

Name: *gridEnable*
Value: t to nil
Set?: Yes
Description: Queries and changes the grid visibility
Equivalent: Color form, Display Group, Grids
Side Effects: Call `axlVisibleUpdate` to update display.

Name: *waiveDRCColor*
Value: 1 to 24
Set?: Yes
Description: Queries and changes the waived DRC color.
Equivalent: Color form, Display Group, Waived DRC.
Side Effects: Call `axlVisibleUpdate` to update display.

Name: *waiveDRCEnable*
Value: t to nil
Set?: Yes
Description: Queries and changes the waived DRC display state.
Equivalent: Color form, Display Group, Waived DRCs
Side Effects: Call `axlVisibleUpdate` to update display.

Name: *schematicBrand*
Value: none concepthd1, capture, scald
Set?: No
Description: Queries current database schematic branding.
Equivalent: See netin in the Allegro PCB Editor dialog.
Side Effects: None.

Name: *cmgrEnabledFlow*
Value: `t` or `nil`
Set?: Yes (only if flag is set)
Description: Reports if board is in constraint manager enabled flow. Only valid in HDL Concept Flow.`t` - Cadence schematic cmgr flow enabled (5 .pst files required)
`nil` - traditional Cadence schematic flow (3 .pst files)
Equivalent: Import Logic Branding shows “Constraint Manager Enabled Flow”
Side Effects: It is not advisable to clear this flag. This interface is provided for those customers who enabled the flow and want to restore the traditional flow. You need to do additional work on the schematic side to clear the option.

Name:*cdszFile*
Value:*t* or *nil*
Set?:*t* or *nil*
Description: Reports if a single `.cdsz` file was used by netrev instead of multiple `.pst` files. If *t*, then `genfeed` format will output a `.cdsz` file.

Name: *schematicDir*
Value: directory path
Set?: Yes
Description: Queries/changes the Cadence schematic directory path. This is the directory location for the Cadence `pst` files. This does not support 3rd party netlist location. If database is unbranded (see above), then the directory location is not stored. Existence of location is not verified.
Equivalent: Import Logic Branding Cadence Tab
Side Effects: none

Name: *testPointFixed*
Value: `t` or `nil`
Set?: Yes
Description: Sets global flag to lock test points. `t` - test points fixed `nil` - test points not fixed
Equivalent: Same as testprep param `"Fixed test points"`
Side Effects: None

Name: *ecsetApplyRipupEtch*
Value: `t` or `nil`
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 Allegro PCB Design XL tools. `t` - ripup etch `nil` - preserve etch
Equivalent: Same as Constraint Manager *Tools — Options — "Rip up etch..."*
Side Effects: None

Name: *maxEtchLayers*
Value: number
Set?: No
Description: Returns maximum number of etch subclasses.
Equivalent: none
Side Effects: none

Name: *dynamicFillMode*
Value: wysiwyg, rough, `nil` (Disabled)
Set?: Yes
Description: Controls filling of dynamic shapes.
Equivalent: shape global param
Side Effects: Disabled in symbol editor.

Name: *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.

Name: *cnsAreaIgnoreUntype*
Value: `t` or `nil`
Set?: Yes
Description: Controls new versus traditional constraint area behavior.

In new mode (`t`), you need to add one of the above properties to the objects to have them use the constraint area. With this mode of operation, you can create constraint areas that just apply to physical or spacing.
Equivalent: cns master dialog - checkbox
Side Effects: None

Name: *maxAttachementSize*
Value: integer value in bytes
Set?: No
Description: Returns the largest attachment size (`axlCreateAttachment`) that may attach to the database.
Equivalent: None.
Side Effects: None.

Name: *activeLayer*

Name: *newFlashMode*
Value: `t` or `nil`
Set?: No
Description: Returns `nil` if board is running old style or `t` if board is running new style flash mode (WYSIWYG negative artwork using fsm files).
Note: WYSIWYG is now smooth in the display.
Equivalent: None
Side Effects: N/A

Name: *dbSize*
Value: int
Set?: No
Description: returns current database size (memory footprint)
Equiv: none
Side Effects: none

Arguments

s_name	Symbol name of control. `nil` returns all possible names.
g_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.

Example 1

```
old = axlDBControl('drcEnable, nil)
```

Sets drc system off.

Example 2

```
current = axlDBControl('ratsnestDistance)
```

Gets current value of pin-2-pin ratsnest.

Example 3

```
listOfNames = axlDBControl(nil)
```

Gets all names supported by this interface.

axlDBIgnoreFixed

Same as axlIgnoreFixed

axlDBSectorSize - Obsolete

```
axlDBSectorSize(
             [f_size]
         )
         ==> nil
```

Description

This is obsolete. It is kept for backwards compatibiltiy. It now calls axlDBTuneSectorSize. 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.

axlIgnoreFixed

```
axlDBIgnoreFixed(
[g_ignore]
) -> t/nil
```

Description

Provides similar functionality to that offered by many Allegro batch programs which allow FIXED property 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 LOCKED or 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`***.***

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`

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)
```

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

)

⇒ t_design/nil
```

Description

Opens a design. The new design replaces the current design. If the current design has unsaved edits, you are asked to confirm before discarding them unless the current design was opened in "r" mode (not supported in this release) or unless *g_mode* is "wf". If you cancel the confirmer, the function returns nil and the current design remains open.

If *t_design* is not provided, you are prompted for the name of the design to open.

If *t_design* does not exist on disk, the standard Allegro PCB Editor Drawing Parameters dialog box appears.

The new design is of the same type as the current design. To reset the design type, specify the appropriate extension with the design name. For example, use a .dra extension to open a file with design type set to SYMBOL, or use a .brd or .mcm extension to open a file with design type set to LAYOUT.

Arguments

t_design	Name of the new design to edit
g_mode	"w" or "wf" (see above)
g_noMru	If t does not update the Most Recently Used file list. (Default is to update.)

Value Returned

t_design	New design name.
nil	No design opened due to incorrect argument.

Note: Functions the same as the Allegro PCB Editor *open* command, except you can set the *t_mode* to "wf" in order to discard the current edits without confirmation.

Since confirmation uses the standard Allegro PCB Editor confirmer, using the "wf" mode is the same as setting the NOCONFIRM environment variable.

This will void all axl dbid handles and clear the selection set.

See Also

axlCurrentDesign, axlKillDesign, axlRenameDesign, axlSaveDesign, axlSetSymbolType, axlRunBatchDBProgram, axlSaveEnable, axlCompileSymbol

axlOpenDesignForBatch

```
axlOpenDesignForBatch(
          ?design t_design
          ?mode 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 confirmer, 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)

Value Returned

`axlOpenDesignForBatch`	`t_design` if opened, or `nil` if error

Note: This command functions the same as the Allegro `edit` command, except the `t_mode` can be set to `"wf"` in order to discard the current edits without confirmation. Since confirmation uses the standard Allegro confirmer, the `"wf"` mode is the same as setting the `NOCONFIRM` environment variable.

This will void all *`axl_dbid`* handles and clear the selection set.

axlRenameDesign

```
axlRenameDesign(
     t_design
)
⇒ t_design/nil
```

Description

Changes the current design name. Has no effect on the existing disk version of the current design. The new current design name will be displayed in the window border and becomes the default name for `axlSaveDesign`.

Arguments

t_design	New current design name.

Value Returned

t_design	New current design name.
nil	Error due to incorrect argument.

See Also

axlOpenDesign

axlSaveDesign

```
axlSaveDesign(
     ?design t_design
     ?mode t_option
     ?noMru g_noMru
)
⇒ 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.

Arguments

t_design	Name for the saved design.
t_option	`"nocheck"` - No database check performed.
g_noMru	If `t` does not update the Most Recently Used file list. (Default is to update.)

Value Returned

t_design	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`

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_drcCount/nil

)

⇒ x_drcCount/nil
```

Description

Changes the units and accuracy of the design. To maintain the current design value, use `nil` for *`t_units`* or *`x_accuracy`*.

When you change units, also change accuracy to maintain adequate precision within the database. Reference the following table to determine the appropriate accuracy.

Units	Min Accuracy	Max Accuracy	Delta
`mils`	0	4	0
`inches`	2	4	3
`microns`	0	2	0
`millimeters`	1	4	3
`centimeters`	2	4	4

- Use the new `DELTA` to decide what the accuracy should be when changing units.

  ```
  new acc = orig acc + (new delta - old delta)
  ```

- This example shows mils to millimeters with a current accuracy of 1:

  ```
  new acc   = orig acc+ (millim delta- mils delta)
       4    = 1       +    3         -    0
  ```

- When changing from one English unit to another or from one Metric unit to another, the default accuracy must match that of the previous unit. For example, if your design uses `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.

Arguments

t_units	`Mils, inches, millimeters, centimeters, or microns.`
x_accuracy	Range is 0 to 4, according to the table shown.

Value Returned

x_drcCount	drc count.
`nil`	Failed to change design units.

Notes:

- This function may take extra time on large designs.
- This function reruns DRC if enabled.

Example 1

```
axlDBChangeDesignUnits("millim" 4)
```

Changes a design from mils/0 to millim/4 (accuracy maintains database precision).

Example 2

```
axlDBChangeDesignUnits(nil 2)
```

Increases accuracy from 0 to 2 and keeps the same units.

axlDBCheck

```
axlDBCheck(
     g_option/lg_options
     [p_file]
)
⇒ (x_errors x_warnings)/nil
```

Description

Runs *dbdoctor* on the current database. By default, no log file is produced. You can specify the `'log` option which writes the standard `dbdoctor.log` file. A port descriptor can be the second argument to write the `dbdoctor` output to an external file.

Arguments

g_option/lg_options

Option	Function
`'general`	Performs a full database check.
`'links`	Performs a full database check.
`'branch`	Performs a full database check.
`'shapes`	Checks shapes (autovoid) ■ May be slow for complicated shapes. ■ Normally fast. ■ Slow on large databases.
`'all`	Performs a full check and fix. Does not perform a shape check.
`'drc`	Performs a batch DRC.
`'log`	Uses the standard `dbdoctor.log` file for output.

p_file 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)
```

axlDBCopyPadstack

```
axlDBCopyPadstack(
     rd_dbid
     l_startEnd
     [g_dontTrim]
)
⇒ o_dbid/nil
```

Description

Creates a new padstack from an existing padstack. The name for the new padstack automatically derives from the existing name by adding a post fix that does not collide with any existing names. The padstack is marked in the database as derived from its starting padstack.

You must provide legal, etch, start and end layers, but `g_dontTrim` must be `t`. If trim is enabled, the new padstack only has pads between the list layers.

Trimming only restricts resulting padstack to between the two indicated layers. If the starting padstack does not already span between the two layers then it will not expand to fill those layers.

Arguments

o_dbid	*dbid* of the padstack to copy from.
lt_startEnd	List of start (*class*/*subclass*) and stop (*class*/*subclass*) layers.
g_dontTrim	Use `t` if you do not want the padstack trimmed, or `nil` to trim the padstack if necessary.

Value Returned

dbid	*dbid* of the new padstack.
`nil`	Failed to create new padstack.

Example

1) To derive an exact copy:

```
newPadId = axlDBCopyPadstack(padId, '("ETCH/TOP" "ETCH/BOTTOM"))
```

2) To derive and trim to only connect from top layer to 2

```
newPadId = axlDBCopyPadstack(padId, '("ETCH/TOP" "ETCH/2") t)
```

axlDBDelLock

```
axlDBDellLock(
     [t_password]

)

⇒ t/nil
```

Description

Deletes a lock on the database. If the database is locked with a password, you must supply the correct password to unlock it.

Arguments

t_password	Password string no longer than 20 characters.

Value Returned

t	Lock is removed or there is no lock to remove.
nil	Failed to remove lock due to an incorrect password.

Example 1

```
axlDBDelLock()
```

Deletes a lock with no password.

Example 2

```
axlDBDelLock("mypassword")
```

Deletes a lock with the password, mypassword.

axlDBGetLock

```
axlDBGetLock(
)
⇒ nil/l_info
```

Description

Returns information about a lock on the database. You retrieve the following information:

t_userName	User login who locked the database.
t_lockDate	Date the database was locked.
t_systemName	System on which the database was locked.
t_export	String representing the current setting for enabling/disabling the ability to export design data to other formats.
t_comments	Comments optionally set by the user who locked the database.

You can also determine whether or not a database is locked as this function returns `nil` when the database is unlocked, and returns a list of information when the database is locked.

Arguments

none

Value Returned

`nil`	Database is unlocked.
l_info	List (*t_userName t_systemName t_export* [*t_comments*]). Database is locked.

axlDBSetLock

```
axlDBSetLock(
     ['password t_password]
     ['comments t_comments]
     ['exports "disabled"/"enabled"]
)
⇒ t/nil
```

Description

Locks the database against future changes. After setting the lock, you must save the database in order to save the lock. After saving the lock, you can no longer save changes to the database.

User is warned when opening a locked database. Attempts to save the database fail with the exceptions of saving the initial lock and uprev.

For simple lock, provide no arguments.

If a nil is provided, returns list of options to this function.

Arguments

t_password	If provided, you need this to unlock the database. If you forget the password, you cannot unlock the database. The password must be 20 characters or fewer. The following characters are not allowed in the password: (\ whitespace) or leading dash (-)
t_comments	Free form comments. You may embed carriage returns (\r) in the string to force a new line in the locking user interface.
'exports	Default is to allow exporting data to other formats. If disabled, the following exports are prevented: techfile write, dump_libraries, and create modules.

Note: Upreving the database from one version to the next ignores the lock flag.

Value Returned

`t`	Locked database.
`nil`	Failed to lock database.

Example 1

```
axlDBSetLock()
```

Locks the database (basic lock).

Example 2

```
axlDBSetLock('comments "I locked the database"
              'exports "disabled")
axlSaveDesign(?design "locked_data")
printf("Locked info %L\n" axlDBGetLock())
```

Locks the database, includes comments, and disables data export.

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

axlPadstackToDisk

```
axlPadstackToDisk(
     [t_padName]
     [t_outPadName]

)

⇒ t/nil
```

Description

Saves a board padstack out to a library.

Arguments

t_padName	Name of the pad to be saved to a library.
t_outPadName	Name of the output pad.

Value Returned

t	Pad is created.
nil	Failed to create pad.

Example 1

```
axlPadstackToDisk()
```

Dumps all the padstacks in the layout.

Example 2

```
axlPadstackToDisk("pad60cir36d")
```

Dumps padstack "pad60cir36d" from the layout as "pad60cir36d.pad".

Example 3

```
axlPadstackToDisk("pad60cir36d" "mypadstack")
```

Dumps padstack "pad60cir36d" from the layout as "mypadstack.pad".

axlTechnologyType

```
axlTechnologyType(
)
⇒ t_technology
```

Description

Returns the type of design technology in use.

Arguments

none

Value Returned

`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

s_trigger	Trigger type. See `axlTriggerSet`.
s_function	Trigger function to remove - the same as you passed to `axlTriggerSet`.

Value Returned

`t`	Trigger removed.
`nil`	Failed to find a trigger by the specified name.

Example

See `axlTriggerSet` for an example.

axlTriggerPrint

```
axlTriggerPrint(
)
⇒ t
```

Description

Debug function that prints what is registered for triggers.

Arguments

`none`

Value Returned

`t`	Always returns `t`.

axlTriggerSet

```
axlTriggerSet(
     s_trigger
     s_function

)

⇒ t/nil

axlTriggerSet(
     nil
     nil

)

⇒ (ls_listOfSupportTriggers)
```

Description

Allows an application to register interest in events that occur in Allegro PCB Editor. When called with both arguments as `nil`, returns a list of supported triggers.

Restrictions

Unless otherwise indicated, in you Skill callback trigger you cannot:

- open, save or close the current database.
- call axlShell.
- call any of the axlEnter functions or any of the Select.
- object functions to request a user pick.

You should restrict any user interaction to blocking dialogs (forms).

Notes

- All trigger functions take a single argument. If you provide a function that does not match this standard, your trigger is **not** called.
- Any processing you do in triggers increases the time to open or save a database. Keep it short! If it must be longer, inform the user regarding the processing delay using `axlUIWPrint`.

- You can register multiple functions for a single trigger, but each registration must have a different function. The order that these functions are called when the trigger occurs is undefined.
- Allegro PCB Editor sends a close trigger when Allegro PCB Editor exits normally. Abnormal exits such as crash, user kill, etc. result in the trigger not being generated.
- You can do user interface work in triggers. You must have the user enter all information before returning from the trigger. Opening a dialog box may involve returning before getting data from the user. To avoid this problem, call `axlUIWBlock` after `axlFormDisplay`. This ensures you do all processing inside the trigger. For correct dialog box display, use the block option in `axlFormCreate`, the *`lt_placement`* parameter.
- The triggers for opening and saving the database are generally not called when the database does temporary saves, for example, `axlRunBatchDBProgram`, reports, or `netrev`.
- You can disable triggering by setting the environment variable *`dbg_noskilltriggers`*. If you suspect that applications running on triggers are causing problems, set this environment variable to help you debug.

Arguments

`s_trigger` Trigger type as shown:

s_trigger Option	Description
`'open (`*`t_database g_existing`*`)`	Called immediately after a database is opened. Passed a list of two items: the name of the database and `t` if existing or `nil` if a new database. Restrictions: You should not open, save, and close the current database.
`'save (`*`t_oldName t_newName`*`)`	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 `*`t_name`*	Called before another database is opened. The single item is the name of the database being discarded.

s_trigger Option	Description
`'exit x_status`	called when program is exiting except if it is an abnormal termination. x_status 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. Restricitions: ■ Do not read or change the database. ■ Do not display dialogs or blocking confirmers.
`'menu t_menuName`	called when a new menu is loaded in the main tool window. Targeted at application code to modify the menu. Restricitions: ■ While a database is active do not change the database. ■ Do not display dialogs or blocking confirmers. ■ Do not call axlUIMenuLoad.
`'xprobe (s_mode lo_dbid)`	Called when object(s) is highlighted or dehilighted s_mode is a symbol which may be highlight or dehighlight and lo_dbid is a list of dbids. This is the same interface used to cross-probe to ConceptHDL/Capture. This is targeted to allow sending messages to external tools. Restricitions: ■ Do not change the database ■ Do not invoke a dialog or blocking confirmer ■ Keep processing at a minimum since this will impact user interactive performance.

s_function	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

Examples

See <*cdsroot*>/share/pcb/examples/skill/trigger for an example.

Example 1

In ilinit, add:

```
procedure( MyTriggerOpen( t_open)
     let( (brd old)
     brd = car(t_open)
     old = cadr(t_open)
     if (old then
          old = "Existing"
     else
          old = "New"
     )
     printf("SKILLCB Open %s database %s\n" old brd)
     t
))
axlTriggerSet('open 'MyTriggerOpen)
```

Shows how to use this with ~/pcbenv/allegro.ilinit to be notified when your user opens a new board. Echoes a print every time a user opens a new database.

Example 2

```
( isCallable('axlTriggerSet) axlTriggerSet('open 'MyTriggerOpen))
```

To be compatible with pre-14.1 software, substitute for `axlTriggerSet` in `allegro.ilinit`.

axlGetActiveLayer

```
axlGetActiveLayer(

)

⇒ t_layer
```

Description

Retrieves active class and subclass of the design.

Note: This function is obsolete and is kept for compatibility reasons. Use `axlDBControl`.

Arguments

None

Value Returned

t_layer	Returns a string denoting the active class and subclass.

Example

```
axlGetActiveLayer()
     ⇒ "MANUFACTURING/PHOTOPLOT_OUTLINE"
```

axlGetActiveTextBlock

```
axlGetActiveTextBlock(
)
⇒ _textBlock
```

Description

Gets the current active text block, equivalent to the status dialog box.

Arguments

None

Value Returned

_textBlock	Returns the current active text block number.

axlSetActiveLayer

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

Description

Sets the active class and subclass of the design.

Note: This function is obsolete and is kept for compatibility reasons. Use `axlDBControl`.

Arguments

t_layer	String with the format: <*class*> **/** <*subclass*>.

Value Returned

`t`	Set the given active class and subclass successfully.
`nil`	Failed to set active the given class and subclass.

Example

```
axlSetActiveLayer( "MANUFACTURING/PHOTOPLOT_OUTLINE")
⇒ t
```

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 paritions exported

See Also

axlDesignType

Exaples

```
axlWFMAnyExported()
```

14

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

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 `axlSetCurrentLayer` function. This function returns a `nil` if you try to set an invalid layer or if you try to create a figure on a layer that does not allow that figure type.

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

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

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

Path Functions

An 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.

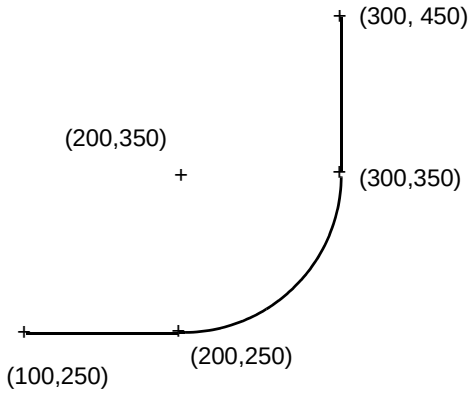
A *`path`* is a set of contiguous arc and single straight line segments. In AXL, you first create a path consisting of the line and arc segments by adding each segment with a separate AXL function, then creating the actual figure using the appropriate `axlDBCreate` function, with the path as one of the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

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

Example

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

l_points	List of *n* vertices, where *n* > 1. If *n* = 1, *r_path* 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 *n* > 1, *r_path* returns with *n*-1 straight-line segments.
f_width	Width for all segments, if any created, between the *l_points*. *f_width* is the default width for all additional segments added to *r_path* using axlPath functions. You can override this default width each time you add a segment using an axlPath function by using a *f_width* argument when you invoke the function.

Value Returned

r_path/nil	Returns the *r_path* handle.

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

Example

See start of the Path Functions on page 734.

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

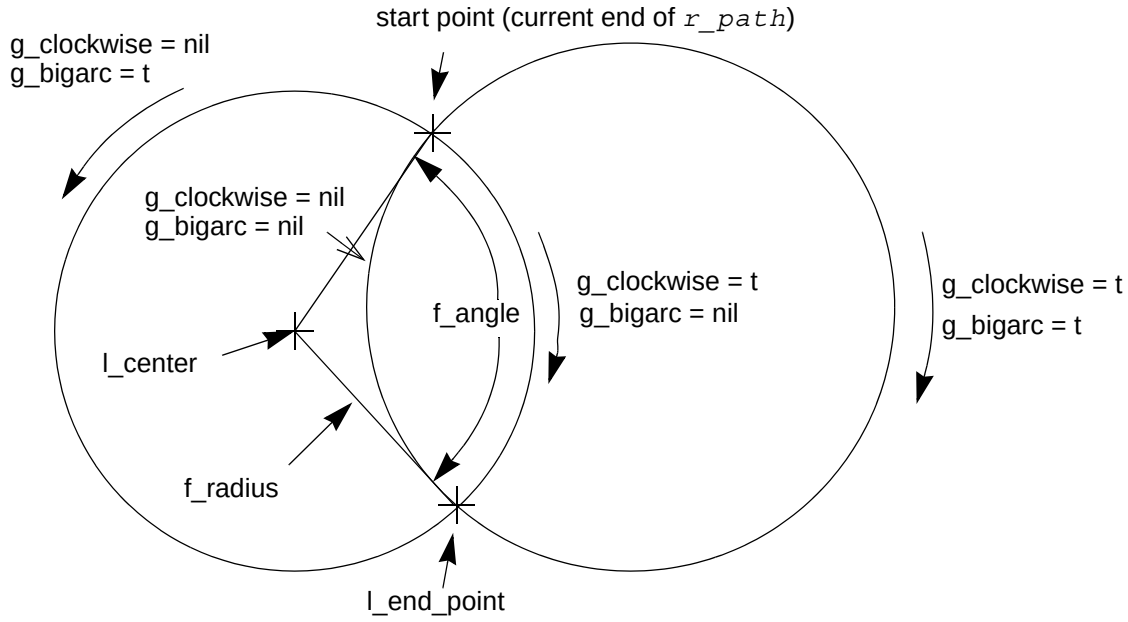
Each of these functions provides a way to construct an arc segment from the current endpoint of *r_path* to the given *l_end_point* in the direction specified by the Boolean *g_clockwise*, as described below and shown in Figure 14-1 on page 740.

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

Arguments

r_path	Handle of an existing *r_path* to receive arc segment.
f_width	Width of an arc segment in user units. Overrides, for this segment only, any width originally given in `axlPathStart`; `nil` = use current width
l_end_point	End point to which an arc is to be constructed. Start point is the last point currently in *r_path* in absolute coordinates.
g_clockwise	Direction to create arc: `t` →create arc clockwise from start to endpoint `nil` →create counterclockwise. Default is counterclockwise (See Figure 14-1 on page 740).
g_bigarc	`axlPathArcRadius`: Create an arc greater than or equal 180 degrees (See Figure 14-1 on page 740).
f_radius	`axlPathArcRadius`: Arc radius in user units.
f_angle	`axlPathArcAngle`: Angle in degrees subtended by arc (See Figure 14-1 on page 740).
l_center	`axlPathArcCenter`: Arc center point in absolute coordinates.

Figure 14-1 Effects of *axlPathArc* Arguments



Value Returned

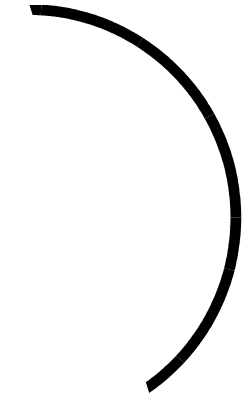
r_path	Current path handle.
nil	Arc path not created.

Example 1

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

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

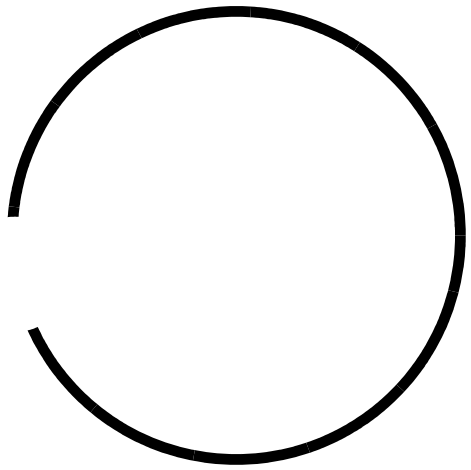
Creates the smaller possible arc:



Example 2

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

Adds a counterclockwise arc subtending 330 degrees.



Example of axlPathArcCenter

See Example on page 735.

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

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

Value Returned

r_path	Path structure following addition of single straight line segment to end of *r_rath* structure.
nil	No line segment added to *r_path* structure.

Example

See start of the Path Functions on page 734.

axlPathGetWidth

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

Description

Gets the default width of an existing path structure.

Arguments

r_path	Handle of an existing path structure.

Value Returned

f_width	Default width of the path structure.
nil	*r_path* is not a path, or is empty.

Example

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

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

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

axlPathSegGetWidth

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

Description

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

Arguments

r_pathSeg	Handle of a segment of a path structure.

Value Returned

f_width	Returns the width of the segment.
nil	*r_pathSeg* is not a segment.

Example

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

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

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

axlPathGetPathSegs

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

Description

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

Arguments

r_path	Handle of an existing path structure.

Value Returned

r_pathList	Returns a list of the segments of the path.
nil	*r_path* is not a path.

Example

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

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

axlPathGetLastPathSeg

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

Description

Gets the last segment of a path structure.

Arguments

r_path	Handle of an existing path structure.

Value Returned

r_pathList	Returns the last segment of the path.
nil	*r_path* is not a path.

Example

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

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

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

axlPathSegGetEndPoint

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

Description

Gets the end point of an existing path structure.

Arguments

r_pathSeg	Handle of a path segment.

Value Returned

l_endPoint	List containing the end point of the path structure.
nil	*r_pathSeg* is not the *dbid* of a path segment, or the structure is empty.

Example

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

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

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

axlPathSegGetArcCenter

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

Description

Gets the center point of a path arc segment.

Arguments

r_pathSeg	Handle of a path arc segment.

Value Returned

l_point	List containing the center coordinate of the arc segment.
nil	Segment is not an arc.

Example

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

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

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

axlPathSegGetArcClockwise

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

Description

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

Arguments

r_pathSeg	Handle of a path segment.

Value Returned

`t`	Segment is clockwise.
`nil`	Segment is counterclockwise.

Example

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

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

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

axlPathStartCircle

```
axlPathStartCircle(
     l_location
     f_width

)

⇒ r_path/nil
```

Description

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

Arguments

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

Value Returned

`r_path`	*`r_path`* with the circle as the only segment.
`nil`	`axlPath` structure not created.

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

Example

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

axlPathOffset

```
axlDB2Path(
          r_path
          xy
         )
         ==> r_path
```

Description

Adds an offset, `xy`, to all points within a `r_path`.

Arguments

`r_path`

`offset` offset xy

Value Returned

new `r_path`

See Also

axlPathStart, axlDB2Path

Examples

Obtain shape outline as a `r_path` and then move if 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 (`od_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 segements, respectively.

Arguments

`od_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)
```

axlDBCreatePath

```
axlDBCreatePath(
     r_path
     [t_layer]
     [t_netName]/['line}
     [o_parent]
     [lo_props]

)

⇒ 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 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 `axlDBAddProp` 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.

Notes:

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

Arguments

`r_path`	Existing path consisting of the straight-line and arc segments previously created by `axlPath` functions
`t_layer`	Layer on which to create a path figure. Default is the active layer.

t_netName	Name of the net to which the path figure is to belong. axlDBCreatePath ignores *t_netName* if *t_netName* is non-nil and *t_layer* is not an etch layer. If the net *t_netName* does not exist, axlDBCreatePath does not create any path, and returns nil.
'line	Changes default path created on an etch layer from a cline to a line.
o_parent	*dbid* of object to be the parent of the path figure. Use the symbol instance or use nil to specify the design. If you attach etch figures to a symbol parent, the figures are not associated with the symbol, and do not move with it.
[lo_props]	Optional list of property name/value pairs. (See axlDBAddProp for format.)

Value Returned

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

See Also

axlDBAddProp

Example

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

; create path on current default layer
axlDBCreatePath(path)

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

;create a line path on top etch layer
```

```
axlDBCreatePath(path "ETCH/TOP" 'line)

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

;create a cline path on top etch layer with properties
proplist = list( `(FILLET t) `(TEARDROP "U1.C17"))
axlDBCreatePath(path "ETCH/TOP" "gnd" nil proplist)
```

axlDBCreateLine

```
axlDBCreateLine(
     l_points
     [f_width]
     [t_layer]
     [t_netname]
     [0_parent]

)

⇒ l_result/nil
```

Description

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

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

All points are in absolute user units.

Arguments

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

Value Returned

`l_result`	List:

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

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

`nil` Nothing is created.

Example

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

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

axlDBCreateCircle

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

Description

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

Arguments

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

Value Returned

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

Example

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

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

Create Shape Interface

You can create shapes using AXL functions as follows:

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

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

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

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

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

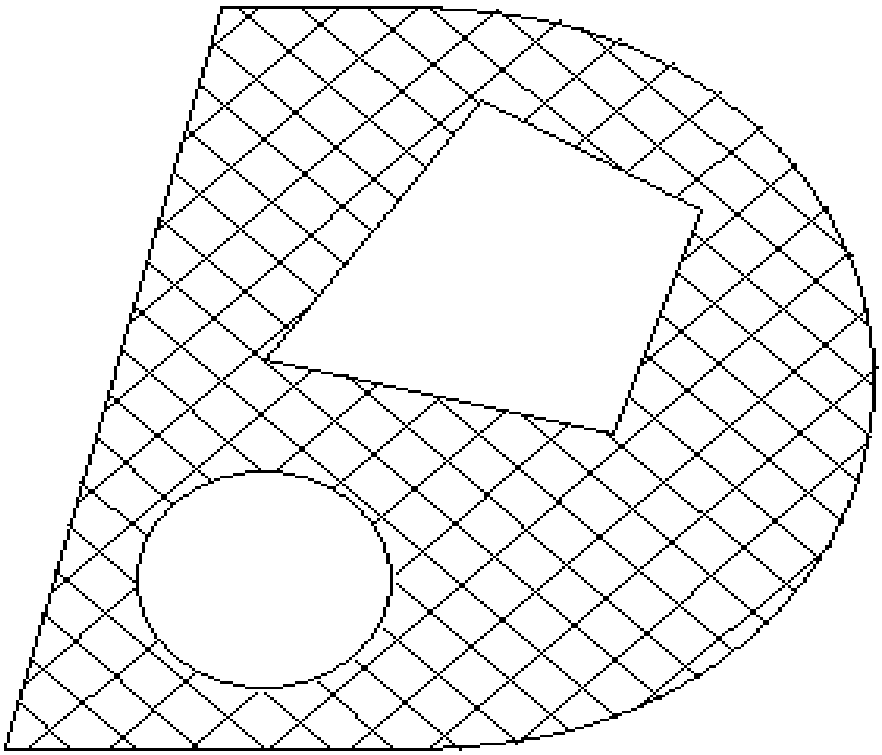
Example

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

```
mypath = axlPathStart( list(1000:1250))
mypath = axlPathLine( mypath, 0.0, 2000:1250)
mypath = axlPathArcCenter(
     mypath, 0.0, 2000:3250, nil, 2000:2250)
mypath = axlPathLine( mypath, 0.0, 1500:3250)
mypath = axlPathLine( mypath, 0.0, 1000:1250)
myfill1 = make_axlFill( ?angle 45.0, ?origin 10:20,
     ?width 50, ?spacing 80)
myfill2 = make_axlFill( ?angle 135.0, ?origin 10:20,
     ?width 5, ?spacing 100)
myfill = list( myfill1 myfill2)
myshape = axlDBCreateOpenShape( mypath, myfill,
     "etch/top", "sclkl")
if( myshape == axlDBActiveShape()
     println( "myshape is the active shape"))
axlDBCreateVoidCircle( myshape, list(1600:1700 300))
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 `"sclkl"` with `axlDBCreateOpenShape`
- Checks that the shape created is the active shape using `axlDBActiveShape` which will print the message
- Creates a circular void and attach it to the shape
- Creates a void shape and attach it to the shape



axlDBCreateOpenShape

```
axlDBCreateOpenShape(
     o_polygon/r_path
     [l_r_fill]
     [t_layer]
     [t_netName/o_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 axlPolyFromDB.)

The shape model uses the open/close model for performance reasons. While adding a shape without voids, you can use axlDBCreateShape, which hides the open and close. While adding voids you should do the following:

```
shape = axlDBCreateOpenShape(...)
... add voids ...
axlDBCreateCloseShape(shape).
```

You can modify an existing shape by using the axlDBOpenShape command, as follows:

```
axlDBOpenShape(shape <new boundary>)
... add or delete voids ...
axlDBCreateCloseShape(shape).
```

Will not allow hole polygons as input. When holes are passed as input, the following warning displays:

```
Invalid polygon id argument -<argument>
```

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 axlDBCreateShape.

fill structure for xhatch shapes is:

l_fill1	A fill_type.

[l_fill2]	(optional) A fill_type. Supplied when more then second xhatch pattern is desired.
[f_outlineWidth]	(optional) Width of outline must be greater then or equal to fill width(s). Specified in design units. Default is current board xhatch width. Only supported for *o_polygon* since outline width for *r_path* should be supplied via the *r_path* defstruct.

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

o_polygon/r_path	The outline as an *r_path* from axlPathXXX data structure or an o_polygon from axlPolyXXX interfaces.
l_r_fill	List of fill structures (*r_fill*) for non solid fill shapes or: t →solid fill nil →unfilled
t_layer	Layer name. nil uses the default active layer.
t_netname	Name of net. Only allowed for shapes being added to etch layers.
o_netdbid	Can use *DBID* of net instead of the netname. Same restrictions apply as for *t_netname*.
o_parent	axl *DBID* of the object to add the shape to. Use the symbol instance, or use nil to specify the design itself.

Value Returned

o_shape	axl *DBID* of the shape. AXL-SKILL does not perform DRC on the shape until you close it using axlDBCreateCloseShape.
nil	No shape created.

Note

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

A path starts at startPoint and a segment is created for each segment in the pathList. If the path does not end at the startPoint, it is considered an error.

A list of o_polygons is not considered valid input. Only a single o_polygon is correct input.

All path segment coordinates are absolute.

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

See Also

axlDBActiveShape, axlDBOpenShape, axlDBCreateVoid, axlShapeDeleteVoids, axlDBCreateCloseShape, axlDBCreateRectangle, axlDBCreateShape, axlDBCreateVoidCircle, and axlShapeAutoVoid

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.il`.

axlDBCreateCloseShape

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

Description

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

Arguments

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

Value Returned

l_result	List: (`car`) *dbid* of the shape created. This *dbid* is not necessarily the same as the *rd_shape dbid* input argument. (`cadr`) `t` if DRCs are created. `nil` if DRCs are not created.
`nil`	Nothing was created.

Example

See Create Shape Interface on page 760 for an example.

axlDBActiveShape

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

Description

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

Arguments

None.

Value Returned

o_shape	*dbid* of the active shape created by axlDBCreateOpenShape.
nil	There is no active shape.

Example

See Create Shape Interface on page 760 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

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

Value Returned

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

Example

See Create Shape Interface on page 760 for an example.

axlDBCreateVoid

```
axlDBCreateVoid(
     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 (axlDBCreateShape) or re-open the shape (axlDBOpenShape) before creating the voids.

Only certain layers, such as ETCH layer, allow voids in a shape. Use axlOK2Void to determine if shape supports voids. While adding multiple voids to an etch shape, first call axlDBOpenShape, for best performance, add the voids then close the shape (axlDBCreateCloseShape).

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

o_shape	*dbid* of the open shape. If this value is `nil`, the command uses the open shape
r_path	Existing path structure created by the `axlPath` functions.

Value Returned

o_polygon	*dbid* of the void created.
`nil`	Error due to calling the function with no open shape.

See Also

axlDBCreateShape, axlDBOpenShape, axlOK2Void, axlDBCreateVoidCircle

Example

- Create Shape Example

  See Create Shape Interface on page 760 for an example.

2) Add to existing shape

  See `dbc_shp_t10` function in `axldbctshp.il` example code.

axlDBCreateShape

```
axlDBCreateShape(
     o_polygon/r_path
     [l_r_fill]
     [t_layer]
     [t_netName]
     [o_parent]

)

⇒ l_result/nil
```

Description

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

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

Arguments

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

`l_r_fill` One of three possible values:

`t` →create shape solid filled

`nil` →create shape unfilled

List of structures specifying crosshatch parameters for creating the shape:

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

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

`t_layer` Layer on which to create the shape.

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

Value Returned

l_result	List: (`car`) *dbid* of the shape created (`cadr`) `t` if DRCs are created. `nil` if DRCs are not created.
`nil`	Nothing is created.

Example

See Create Shape Interface on page 760 for an example.

axlDBCreateRectangle

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

Description

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

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

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

- Net does not exist.
- Attempt to create a filled rectangle on an 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 axlDBCreateSymbolSkeleton for notes about restrictions on shapes that are part of symbol definitions.

Arguments

l_bBox	Bounding box of the rectangle: Lower left and upper right corners of rectangle
g_fill	If `t` then the fill is solid. If `nil` (default) then the rectangle is unfilled.
t_layer	Layer to which to add the rectangle. Default is the active layer.

t_netname	Name of net to which the rectangle is to belong. This argument is meaningful only if the rectangle is being added on an Etch layer.
o_parent	*dbid* of object of which the rectangle is to be a part. Use either the *dbid* of a symbol instance or use `nil` to specify the design itself.

Value Returned

l_result	List: (`car`) rectangle *dbid* (`cadr`) `t` if DRCs are created. `nil` if DRCs are not created.
`nil`	Nothing is created.

Example

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

axlDBCreateRectangle(list(200:200 400:300) t)

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

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

Nonpath DBCreate Functions

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

axlCreateBondFinger

```
axlCreateBondFinger(
             parentSymbol
             fingerName
             list(fingerLocation fingerRotation fingerPadstack)
             list(placementStyle ewlLength fingerSnap fingerAlign)
          )
          ==> dbid/nil
```

Descrption

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

parentSymbol	dbid of the symbol (generally a die) with which this finger should be associated when performingoperations like a move or delete.
fingerName	The optional parameter that specifies the name of the bond finger, as stored in the BOND_PAD property.
fingerLocation Rotation Padstack	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.
placementStyle/ewlLength/ fingerSnap/fingerAlign	The placement data for the bond finger being created, as follows:

placementStyle	String value in the following list: ■ Orthogonal ■ Equal Wire Length ■ On Path ■ Free Placement
ewlLength	Length value for Equal Wire Length style, which represents the desired length of the wire.
fingerSnap	String value in the following list: ■ Center of Finger ■ Finger Origin ■ Near End ■ Far End ■ Nearest Point ■ Farthest Point
fingerAlign	String value in the following list: ■ Aligned with Wire ■ Orthogonal to Die Side ■ Orthogonal to Guide ■ Pivoting Ortho to Guide ■ Average Wire Angle ■ Constant Angle ■ Match CW Neighbor ■ Match CCW Neighbor

Value Returned

- `dbid` of newly created bond finger if successful.
- `nil` if an error occured (message printed to status window).

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).

Value Returned

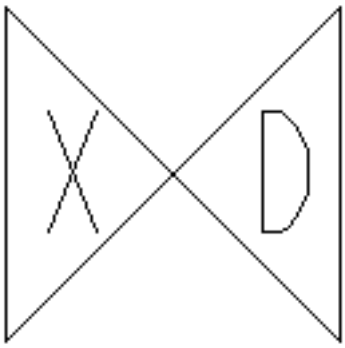
- dbid of newly created bond wire if successful.
- nil if an error occured (message printed to status window).

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 exepctValue. 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. Note constraint and expectValue are implemented as properties on the external drc *dbid* whose names are EXTERNAL_VIOLATION_DESCRIPTION and EXTERNAL_DRC_VALUE.

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

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.

Arguments

t_constraint	Name of the violated constraint. String contains the type of constraint and the required value and comparison.
lt_constraint	Alternative method it is a list of (t_constraint t_expectValue)
l_anchor_point	Coordinate of the DRC marker.
t_layer	Layer of the DRC marker.
lo_dbid	Optional list of the objects that caused the DRC (maximum of two).
l_secondPoint	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.
t_actualValue	Actual value that caused the DRC.

Value Returned

l_result	List: (car) *dbid* of the DRC created (always only one) (cadr) `t` (always)
`nil`	Nothing is created.

Example

Creates a user defined DRC marker at x1500, y1800 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:

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

New method for better `show element` and `reports` command behavior:

```
axlDBCreateExternalDRC('("My Spacing Line to Pin" "12")
```

```
1500:1900 "top", nil nil "10 MILS")
```

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

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

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

axlDBCreatePadStack

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

Description

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

Note: This function is not available in the Symbol Editor.

Defstructs used to create padstack

Drill *(r_drill)* use make_axlPadStackPad. Elements are:

- fixed = *(t/nil)* internal fixed flag
- uvia = *(t/nil)* if padstack is of type bbvia set as sub-type micro-via
- keepout = *(t/nil)* if padstack is used for mechanical pins, its anti-pads have been set-up to act as route keepouts. If this pad is used for logical pins then this option is ignored.
- drillDiameter = *(float)* drill hole size setting
- figure = *(symbol)* the drill figure. Allowed symbols are NULL, CIRCLE, SQUARE, HEXAGON, HEXAGON_X, HEXAGON_Y, OCTAGON, CROSS, DIAMOND, TRIANGLE, OBLONG_X, OBLONG_Y, RECTANGLE. Note nil is treated as NULL.
- figureSize = *((f_width f_height))* size of drill figure
- offset = *((f_x f_y))* drill hole offset
- plating = *(symbol)* plate status of drill hole
  Symbols are: NON_PLATED, OPTIONAL, nil
- drillChars = *(string)* drill characters identifier. Up to two characters are allowed.

`multiDrillData` = list for multiple drill data which is:
`( (x_num_rows nx_um_columns f_clearance_x [f_clearance_y ["staggered"]]) )`
data type is *`(int int float [float])`*

`holeType` = *`(symbol)`* the hole type. Allowed symbols are CIRCLE_DRILL, OVAL_SLOT, RECTANGLE_SLOT.
nil is treated as CIRCLE_DRILL.

`slotSize` = *`( (f_width f_height) )`* size of slot hole

`holeTolerance` = *`(f_pos f_neg) )`* +/- hole tolerance

`drillNonStandard` = *`(symbol)`* non-standard drill hole. Allowed symbols are LASER_DRILL, PLASMA_DRILL, PUNCH_DRILL, PHOTO_DRILL, COND_INK_DRILL, OTHER_DRILL.

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

`layer` = *`(string)`* etch layer name (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.

`type` = *`(symbol)`* pad type
Allowed symbols: ANTIPAD, THERMAL or REGULAR. nil is treated as REGULAR.

`flash` = *`(string)`* the pad aperture flash name. Reference a flash, shape symbol name or nil for no flash.

`figure` = *`(symbol)`* the pad figure
Allowed symbols: NULL, CIRCLE, SQUARE, FLASH, RECTANGLE, OBLONG_X, OBLONG_Y, OCTAGON. If NULL, checks the figureSize 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 symbolcalled `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 via PADPATH. Also, the ?figureSize attribute must be the extents of the symbol or larger.

`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

Arguments

`t_name` Padstack name.

`r_drill` Drill hole data for the padstack.

Note: As with all SKILL defstructs, use the constructor function `make_axlPadStackPad` to create instances of `axlPadStackPad`. See Create Shape Interface on page 760 for an example.

`l_pad` Pad definition data for the padstack.

Note: As with all SKILL defstructs, use the constructor function `make_axlPadStackPad` to create instances of `axlPadStackPad`. See Create Shape Interface on page 760 for an example.

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

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

Value Returned

l_result	*dbid* of the padstack created.
nil	Nothing is created.

Examples

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

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

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

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

To use a shape symbol “myshape”, do

```
pad = make_axlPadStackPad(?layer "TOP",
            ?type `REGULAR, ?figure "myshape"
            ?figureSize 80:80) pad_list)
```

To use a flash symbol “myflash”, do

```
pad = make_axlPadStackPad(?layer "TOP",
            ?type 'THERMAL,
            ?figure 'FLASH ?flash "myflash"
            ?figureSize 80:80 ) pad_list)
```

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

t_padstack — 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.

`o_padstackDbid` — a padstack dbid

l_anchorPoint — Layout coordinates of the location to add the pin.

r_pinText — Pin number text structure:

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

This requires the `axlTextOrientation` structure:

```
defstruct axlTextOrientation
     ;;(r_textOrientation) - description of
     ;; the orientation of text
textBlock     ;string - text block name
```

```
rotation      ;rotation in floatnum degrees
mirrored      ;t-->mirrored, nil --> not mirrored
     ;'GEOMETRY --> only geometry is mirrored
justify) ;"left", "center", "right"
```

Note: As with all SKILL defstructs, use constructor functions `make_axlPinText` to create instances of `axlPinText` and `make_axlTextOrientation` for `axlTextOrientation`. See Create Shape Interface on page 760 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

l_result — List:

(`car`) *dbid* of the pin

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

`nil` — Nothing is created.

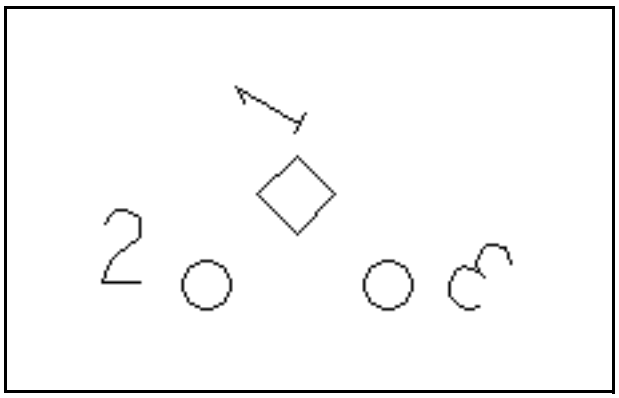
Example

- 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")
        mypin = make_axlPinText(?number "1",
                              ?offset 0:75, ?text mytext)
        axlDBCreatePin( "pad1" 0:0 mypin 45.0)
    mytext->justify = "left"
        mytext->rotation = 0.0
        mypin->number = 2
        mypin->offset = -125:0
        axlDBCreatePin( "pad0" -100:-100 mypin)
    mytext->rotation = -45.0
        mytext->justify = "right"
        mytext->mirror = t
        mypin->number = 3
        mypin->offset = 50:0
```

```
    axlDBCreatePin( "pad0" 100:-100 mypin)
mypin->mytext = nil
    axlDBCreatePin( "pad0" 100:100 mypin)
```

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]

)

⇒ l_result/nil
```

or

```
axlDBCreateSymbol(
     l_symbolData
     l_anchorPoint
     [g_mirror]
     [f_rotation]

)

⇒ l_result/nil
```

Description

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

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

Arguments

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

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

l_symbolData

If this is the first argument, the function looks for the symbol, symbol type, and refdes specified by this structure. *l_symbolData* is a list (*t_symbolName* [[*t_symbolType* [*t_refdes*]]), where:

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

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

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

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

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

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

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

t_refdes

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

l_symbolData

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

l_anchorPoint

Layout coordinates specifying where to create the symbol.

g_mirror

`t` →create symbol mirrored.

`'GEOMETRY` →only geometry is mirrored.

`nil` →create unmirrored.

f_rotation

Rotation of the symbol in degrees.

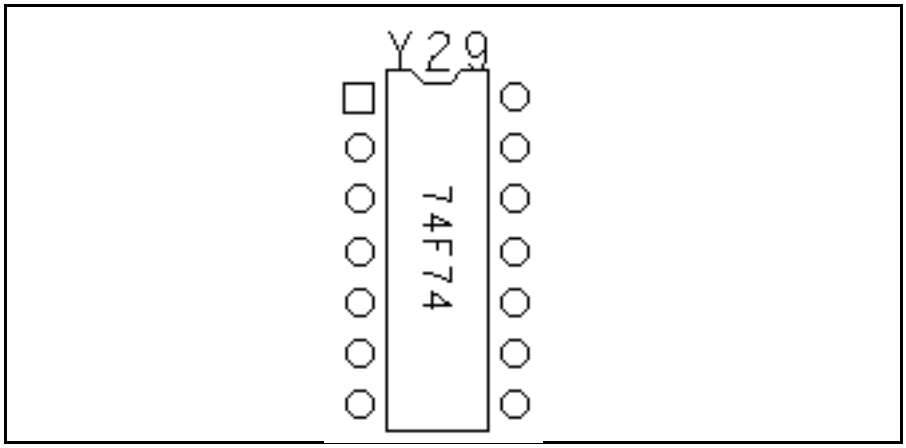
Value Returned

axlDBCreateSymbol	List:
	(car) axl *DBID* of the symbol created
	(cadr) t if DRCs are created. nil if DRCs are not created.
nil	Nothing is created.

Example

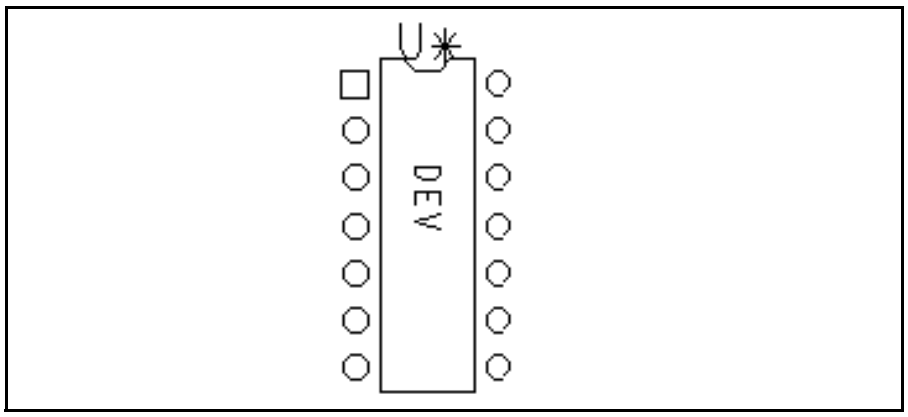
```
axlDBCreateSymbol("y29", 5600:4600)
     =>(dbid:423143 nil)
```

Creates a symbol with the assigned refdes.




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

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



axlDBCreateSymbolSkeleton

```
axlDBCreateSymbolSkeleton(
     t_refdes
     l_anchorPoint
     g_mirror
     f_rotation
     l_pinData

)

⇒ l_result/nil
```

or

```
axlDBCreateSymbolSkeleton(
     l_symbolData
     l_anchorPoint
     g_mirror
     f_rotation
     l_pinData

)

⇒ l_result/nil
```

Description

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

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

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

t_refdes

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

l_symbolData

If this is the first argument, the function looks for the symbol, symbol type, and refdes specified by this structure. *l_symbolData* is a list (*t_symbolName* [[*t_symbolType* [*t_refdes*]]), where:

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

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

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

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

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 symobl's origin.

g_mirror t →create symbol mirrored.

'GEOMETRY →only geometry is mirrored.

nil →create unmirrored.

f_rotation Rotation angle of the symbol in degrees.

nil →0.0.

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

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

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

Value Returned

axlDBCreateSkeleton List:

(car) axl *DBID* of the symbol created

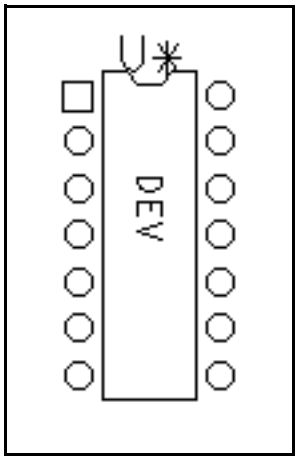
(cadr) t if DRCs are created. nil if DRCs are not created.

`nil` Nothing is created.

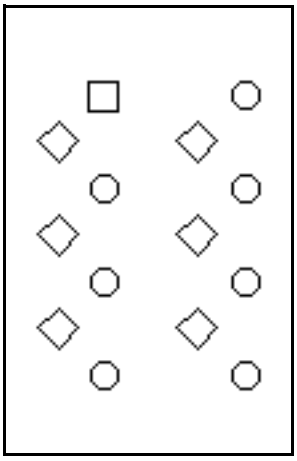
Example

```
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))

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

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



Library symbol



Skeleton symbol created by code sample above

axlDBCreateText

```
axlDBCreateText(
     t_text
     l_anchorPoint
     r_textOrientation
     [t_layer]
     [o_attach]

)

⇒ l_result/nil
```

Description

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

Arguments

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.

r_textOrientation

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 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

l_result	Otherwise the function returns a list: (car) list of text *DBIDs* created, one for each line of text input (cadr) t if DRCs are created. Otherwise the function returns nil.
nil	Nothing is created.

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

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

Value Returned

l_result	List:

(`car`) *`DBID`* of the via created.

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

nil — Nothing is created.

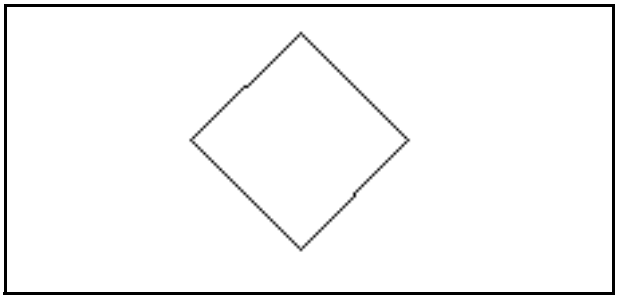
Note

`axlDBCreateVia` cannot create a test point. You have to create testpoints by using the `axlTestPoint` function.

Example

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

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



axlDBCreateSymbolAutosilk

```
axlDBCreateSymbolAutosilk
     o_symbol

)

⇒ t/nil
```

Description

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

Arguments

o_symbol	*dbid* of the symbol.

Value Returned

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

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 to, etc. through the wirebond

tools.

Arguments

`r_path`	Existing path consisting of the straight-line and arc segments previously created by axlPath functions

Value Returned

- dbid of newly created guide path if successful.
- nil if an error occured (message printed to status window).

Property Functions

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

axlDBCreatePropDictEntry

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

Description

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

Arguments

t_name	Name of the property. Must be different from all other property names in the design, both Allegro PCB Editor pre-defined and user-defined property names.
t_type	Data type of the property value. See the AXL-Skill FSpec Property Database Types attached. (Use `axlDBGetPropDictEntry (nil)` to get a list of valid objects.) If only a single object type is allowed, then attached (see the AXL-Skill FSpec). Legal values are: Typical: BOOLEAN, INTEGER, REAL, STRING, and DESIGN_UNITS. Other supported types are: ALTITUDE CAPACITANCE

DISTANCE

ELEC_CONDUCTIVITY

ENUM

FAILURE_RATE

IMPEDANCE

INDUCTANCE

LAYER_THICKNESS

NAME

NOISE VOLTAGE

PERCENTAGE

PROP_DELAY

RESISTANCE

TEMPERATURE

THERM_CONDUCTANCE

THERM_CONDUCTIVITY

THERM_RESISTANCE

VOLTAGE

VELOCITY

lt_objects List of strings representing the object types to which this property can be added. (Use `axlDBGetPropDictEntry (nil)` 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.

The allowed object types are:

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

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

t_units — A text string so be used with data types (*t_type*) without units, such as `STRING`, `INTEGER`, or `REAL`.

g_hidden — *t* property is hidden from the user. Hidden properties are not shown in any Allegro UI like Constraint Manager, Show Element or Propety 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 `extracta`.

Value Returned

o_propDictEntry — *DBID* of the property dictionary entry created.

`nil` — Property not created.

Example

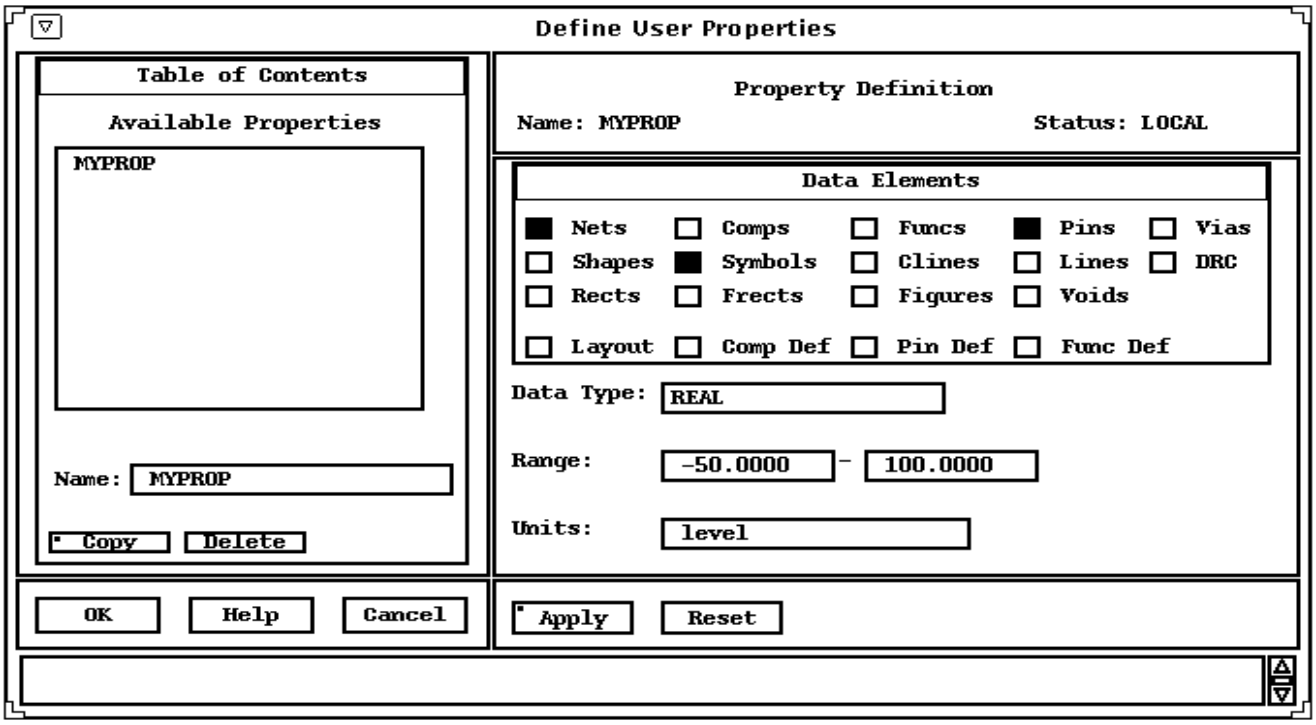
```
axlDBCreatePropDictEntry( "myprop", "real", list( "pins" "nets" "symbols"),
list( -50. 100), "level")
propDict:2421543
```

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

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.

axlDBAddProp

```
axlDBAddProp(
     lo_attach
     ll_name_value

)

⇒ l_result/nil
```

Description

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

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

Arguments

lo_attach
: List of Allegro PCB Editor object dbids to which to add the property/value combinations listed in *ll_name_value*. 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 as lists. If the car of this list is not a list, then axlDBAddProp treats *ll_name_value* as a single name-value list. The car of each name-value pair is the property name as a string. The cadr of the name-value list is the property value. It is either a string with or without units included, or a simple value (fixed or floating). If the value does not include units explicitly, then axlDBAddProp uses the units specified in the system units.dat file.

  axlDBAddProp ignores the property-value if the property data type is BOOLEAN.

Value Returned

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

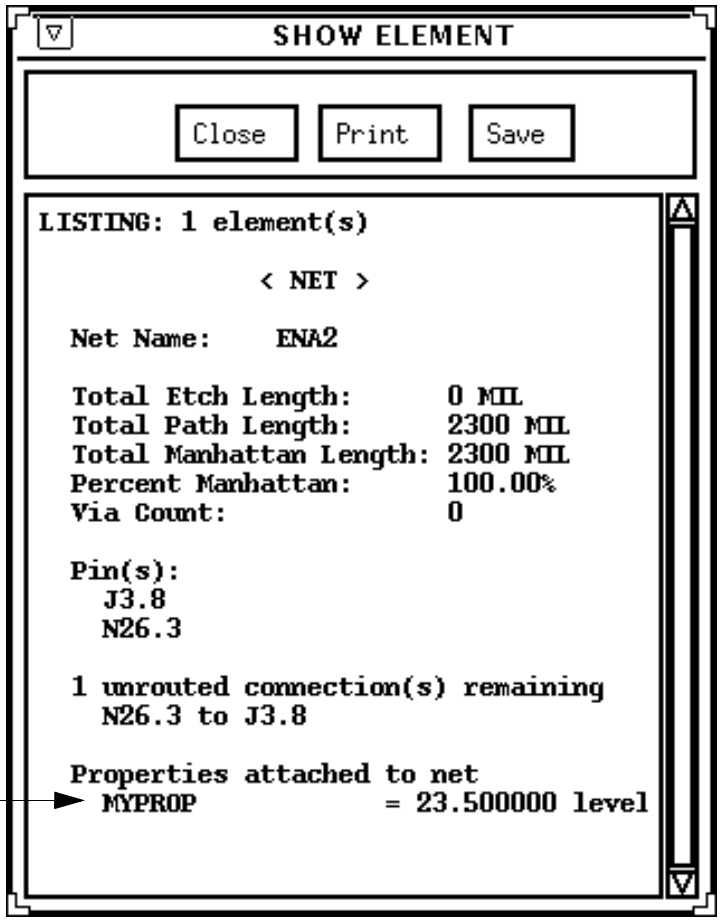
See Also

axlDBDeleteProp, axlDBCreatePropDictEntry axlDBGetPropDictEntry axlDBGetProperties, axlDBDeletePropAll, axlDBDeletePropDictEntry, and axlDBGetPropDict

Example

see axlDBDeleteProp

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



Load 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

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

Value Returned

o_dbid	*dbid* of padstack loaded.
nil	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 load sthe symbol into the database. In the symbol editor, this can only be used for shape and mechanical symbols for use with padstacks.

if a symol definition is not is use (dbid->instance is nil) then the definition is deleted. This deletion of unused symbols occurs during save drawing, refresh symbol place manual amoung other place. This means the database is saved as part of axlRunBatchDBProgram then the unused symdefs will be deleted.

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

t_symkind	PACKAGE, MECHANICAL, FORMAT , SHAPE or FLASH (case insensitive)
t_symName	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

dbid	Of symbol definition

nil	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")
```

15

Database Group Functions

Overview

This chapter describes the AXL-SKILL Database Group functions.

axlDBAddGroupObjects

```
axlDBAddGroupObjects(
     o_group
     lo_members
     )
```

⇒ `t/nil`

Description

Adds the database objects specified in the new members list to a group. All restrictions and disclaimers specified in `axlDBCreateGroup` also apply for this procedure.

Arguments

o_group	*dbid* of the group to receive new members.
lo_members	List of *dbid*'s specifying the new group members. Database objects already in the group are silently ignored (giving a return value of `t`.) A single *dbid* 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 `axlDBCreateGroup`.

See Also

axlDBCreateGroup

axlDBCreateGroup

```
axlDBCreateGroup(
     t_name
     t_type
     lo_groupMembers
     )
```

⇒ *o_dbid*/nil

Description

Creates a new group database object with members specified by *lo_groupMembers*.

Arguments

t_name	String providing the group name. If name is in use by an existing group, this function fails and returns `nil`.
t_type	String defining the group type. Legal values include: "`generic`"
lo_members	List of AXL *dbids* defining the group members. Duplicate *dbid* entries are silently ignored. An object is added to a group only once. A single *dbid* can be substituted for a list. If certain restrictions on the group members are violated, this function fails and returns `nil`. The following are restrictions for all types of groups: ■Only *dbid*s of the following *objType* are allowed: ❑group ❑component ❑symbol ❑net ❑path ❑via ❑shape ❑polygon

- pin
- text

- Circular group relationships cannot be formed. For example, Group A can not be added as a member of Group B if Group B is directly or indirectly a member of Group A.

Value Returned

o_dbid	*dbid* of the newly formed group.

See Also

axlDBAddGroupObjects, axlDBDisbandGroup, axlDBRemoveGroupObjects

Example

```
groupMembers = axlGetSelSet()
group_dbid = axlDBCreateGroup("my_group" "generic" groupMembers)
```

Note: The order of the group members provided when you access the *groupMembers* property may vary from the order provided in *lo_groupMembers*.

axlDBDisbandGroup

```
axlDBDisbandGroup(
     o_group
     )

⇒ t/nil
```

Description

Disbands the database group you specify with the *o_group* argument, thereby immediately removing the group. Members of the group are not deleted.

Arguments

o_group	*dbid* of the group to be deleted.

Value Returned

t	Group disbanded.
nil	Group could not be disbanded due to an invalid argument, for example, the *dbid* not being for a valid group.

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 diffpair group.

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_promoteToNet`	Promote object to its net/xnet and perform test on that object. Use this 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.

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

- diffpair; 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")
```

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

o_group	Group *dbid*.
lo_members	List of database objects to be removed from the group. A single *dbid* can be substituted for a list.

Value Returned

t	One or more objects removed from the group.
nil	*lo_members* contained no *dbids* of objects which could be removed from the group.

Notes:

- If a group is left with no members, the group is tagged for deletion, but is not removed immediately.
- You do not need to explicitly remove objects from a group before deleting the object with axlDeleteObject. Deleting an object removes it from all groups to which it belongs.

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
- xnets
- diffpairs
- busses

See netclass discussion in axlNetClassCreate. 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

o_netclassdbid:	dbid of a netclass group
t_netclassName:	name of a netclass group
o_dbid:	legal database dbid to add to netclass
lo_dbid:	list of legal database dbids to add to netclass

Value Returned

t	added elements
nil	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(mg  nc)
```

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 netclasses in the database via: axlDBGetDesign()->netclass. axlNetClassGet reports if an object is a member of a netclass either directly or via the logic hierarchy. Netclasses can be constrainted by assigning csets via the PHYSICAL_CONSTRAINT_SET, SPACING_CONSTRAINT_SET or ELECTRICAL_CONSTRAINT_SET. Same Net constraints shares the same domain with the SPACING_CONSTRAINT_SET.

Arguments

t_name	name of netclass group (changed to upper case)
g_domain	netclass domain can be 'spacing, 'physical, 'electrical or 'all
lg_domain	list of netclass domains

Value Returns

nil:	error or netclass with that name exists
o_dbid:	dbid of group

See Also

axlNetClassDelete, axlNetClassAdd, axlNetClassRemove, axlNetClassGet

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

o_netclassdbid:	dbid of a net class group
t_netclassName:	name of a net class group
lg_netclassdbid:	list of net class groups (dbids or names)

Value Returned

t:	net class group deleted
nil:	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_hierarchal is nil, returns object's netclass if a direct member. If g_hierarchal=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

o_dbid:	dbid may be net, xnet, diffpair or bus
s_domain:	netclass domain; spacing, physical or electrical
g_hierarchal	

Value Returned

o_netclass	dbid of netclass
nil	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 casses

```
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.

Note: Using dbids is faster then using names.

Arguments

o_netclassdbid:	dbid of a netclass group
t_netclassName:	name of a netclass group
o_dbid:	legal database dbid to remove from group
lo_dbid:	list of legal database dbids to remove from group

Value Returned

t	removed elements
nil	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(mg  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

o_regiondbid:	dbid of a region group
t_regionName:	name of a region group
o_dbid:	legal database dbid to add to region
lo_dbid:	list of legal database dbids to add to region

Value Returned

t	added elements
nil	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 modifing regions you may wish to wrap all the calls with the axlDBCloak comman.

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 rgion called "BGA"

```
nc = axlRegionCreate("BGA")
```

See Also

axlRegionDelete, axlRegionAdd, axlRegionRemove, axlDBCreateShape, axlDBCreateRectangle

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

o_regiondbid:	dbid of a region group
t_regionName:	name of a region group
lg_regiondbid:	list of region groups (dbids or names)

Value Returned

t:	net class group deleted
nil:	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

o_regiondbid:	dbid of a region group
t_regionName:	name of a region group
o_dbid:	legal database shapes to remove from group
lo_dbid:	list of legal database shapes to remove from group

Value Returned

t	removed elements
nil	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)
```

16

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. Attachment data may be specified as a string or as a file name.

`axlDBControl('maxAttachmentSize)` returns the maximum size of data that can attach to the database.

Arguments

t_attachmentId	Id or name of the attachment to retrieve. Can be up to 31 characters in length.
t_passwd	Password for this attachment. Can be up to 31 characters in length. If no password is desired this may be `nil`.
x_revision	Revision number of the attachment. If `nil`, the revision number is set to zero.
s_dataFormat	Indicates the format of the *t_data* argument. If *s_dataFormat* is `'string`, the value of the *t_data* argument is used for the attachment data. If *s_dataFormat* is `'file`, then the *t_dataString* argument is interpreted as a file name from which the attachment data is read.
t_data	String of ascii characters representing the attachment data. May represent the data itself or the name of the file from which to read the data, depending on the value of the *s_dataFormat* argument.

Value Returned

o_attachment	AXL id for the new attachment, which can then be queried using the right arrow (->) operator.
nil	Failed to create an attachment due to incorrect arguments.

Note: Once an attachment is password protected it needs to be deleted, then re-added to remove or change the password protection.

For additional information, see: `axlIsAttachment, axlGetAttachment, axlGetAllAttachmentNames, axlSetAttachment, axlDeleteAttachment, axlDBControl`

axlDeleteAttachment

```
axlDeleteAttachment(
     t_attachmentId
     [t_passwd]
     )

⇒ t/nil
```

Description

Deletes the given attachment. If the attachment is password protected, the correct password must be given.

Arguments

t_attachmentId	Id or name of the attachment to delete.
t_passwd	Password for this attachment.

Value Returned

t	Attachment deleted.
nil	Attachment not deleted.

axlGetAllAttachmentNames

```
axlGetAllAttachmentNames(
     )
⇒ l_attachment/nil
```

Description

Returns a list of the ids for all database attachments in the Allegro PCB Editor database. If no attachments are present, then `nil` is returned. You can retrieve the attachments using the `axlGetAttachment()` function.

Arguments

none

Value Returned

`l_attachment`	List of attachment ids.
`nil`	No attachments exist in the database.

axlGetAttachment

```
axlGetAttachment(
     t_attachmentId
     [s_dataFormat]
     )

⇒ o_attachment/nil
```

Description

Returns the database attachment with the given id. If the attachment exists, an *attachment record* is returned containing information about the attachment. The data is in the format specified by the *s_dataFormat* argument. If `'file` format, then the *data* attribute contains a temporary file name to which the data was written. If `'string`, then the *data* attribute contains the attachment data itself. If the *s_dataFormat* argument is omitted or is `nil`, then the *data* attribute is `nil`.

The attachment record has the following attributes:

Name	Type	Set?	Description
objType	string	NO	Is always "`attachment`".
id	string	NO	Id (name) of the attachment.
password	boolean	NO	`t`/`nil` - Indicates if the attachment is password protected.
timeStamp	integer	NO	Indicates the time last modified in seconds.
revision	integer	YES	User defined revision number for the attachment data.
dataFormat	symbol	YES	Indicates the format of the data stored in the "data" attribute and is one of `'file`, `'string`, or `nil` (in which case the data is not displayed.)
data	string	YES	Attachment data. May be a file name, the data itself, or `nil` depending on the value of the *dataFormat* attribute.
timeStamp	integer	NO	Indicates the size of the attachment.

Arguments

t_attachmentId	Id or name of the attachment to retrieve. Can be up to 31 characters in length.
s_dataFormat	Format in which the attachment data is stored in the "*data*" attribute. Must be `'string`, `'file`, or `nil`.

Value Returned

o_attachment	AXL id for the attachment structure which can be queried using the right arrow (->) operator.
`nil`	Attachment does not exist.

Example

```
attachment = axlGetAttachment("attachmentOne" 'file)
⇒attachment:attachmentOne
```

axlIsAttachment

```
axlIsAttachment(
     o_attachment
     )
```

⇒ `t/nil`

Description

Determines if the given object is an AXL attachment.

Arguments

o_attachment	Object to check.

Value Returned

`t`	Object is an attachment.
`nil`	Object is not an attachment.

axlSetAttachment

```
axlSetAttachment(
     o_attachment
     [t_password]
     )

⇒ o_attachment/nil
```

Description

Modifies an existing 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 `axlGetAllAttachments` function. The attachment revision number and the attachment data may both be modified.

Format of the data is determined by the *`dataFormat`* attribute structure, which may be set by the user. If "*`dataFormat`*" is `'string`, then the value of the *`data`* attribute is used for the new attachment data. If "*`dataFormat`*" is `'file`, then the value of the *`data`* attribute is a file name from which the attachment data is read.

If the existing attachment is password protected, you must provide the correct password or the function fails.

Arguments

`o_attachment`	AXL id of the existing attachment to be modified. The *`revision`*, *`dataFormat`*, and *`data`* attributes may all be set to new values by the user.
`t_password`	Password for the given existing attachment. If this does not match the password of the existing attribute, the attachment update fails. If the existing attachment is not password protected, you may omit this.

Value Returned

`o_attachment`	Id of the modified attachment.
`nil`	Failed to modify the attachment.

Note: Once an attachment is password protected, to remove or change the password protection you must delete and then re-add the attachment.

17

Database Transaction Functions

Overview

This chapter describes the AXL-SKILL Database Transaction functions.

axlDBCloak

```
axlDBCloak(
     g_func
     [g_mode]
     )

⇒ g_return
```

Description

Improves performance and program memory use while modifying many items in the database. You use `axlDBCloak` to update many etch or package symbols in batch mode. Works like SKILL's `eval` function. You pass it a function and its arguments using the following format:

```
axlDBCloak ('MyFunc( myargs) )
```

You can use `axlDBCloak` to do the following:

- Batch any net based DRC updates.
- Batch connectivity update.
- Optionally batch dynamic shape updates if *g_mode* is `'shape`.
- Incorporate an errset around your function so any SKILL errors thrown are caught by `axlDBCloak`.

Do not use `axlDBCloak` in these circumstances:

- If you are adding or deleting non-connectivity database items (for example, loading many lines to a manufacturing layer)
- If you need to interact with the user. Since connectivity is not updated, do not use the `axlEnter`*XXX* functions. Instead, get the information from the user first, then do the cloak update.
- If you are reading the database, using cloak does not help and may actually slow performance.
- If making a single change, using Cloak slows performance.

Note: Using Cloak sets any database ids to `nil`.

The `'shape` *g_mode* option improves performance if changes effect any dynamic shapes on the design. Set this if your changes effect `ETCH` layers.

Tip

For effective debugging, first call your function directly from the top level function, then wrap in the cloak call.

Arguments

g_func	Function with any of its arguments.
s_mode	`nil` or `'shape`

Value Returned

Returns what *g_func* returns.

Example

```
procedure( DeleteSymbols()
let( (listOfSymbols)

     listOfSymbols = axlDBGetDesign()->symbols
     when( listOfSymbols
          axlDBCloak( 'DeleteDoit(listOfSymbols) 'shape ))
          axlShell("cputime stop"
     ))
procedure( DeleteDoit(listOfDatabaseObjects )

     foreach(c_item listOfDatabaseObjects
          printf("REFDES %s\n", c_item->refdes)
          axlDeleteObject(c_item)
     )
     nil
)
```

Deletes all placed symbols in the database.

axlDBTransactionCommit

```
axlDBTransactionCommit(
     x_mark
     )
```

⇒ `t/nil`

Description

Commits a database transaction from the last transaction mark.

Arguments

x_mark	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

x_mark	Database transaction mark returned from `axlDBTransactionStart`.

Value Returned

`t`	Mark written in the database.
`nil`	No mark written in the database.

Example

See `axlDBTransactionStart()` for an example.

axlDBTransactionOops

```
axlDBTransactionOops(
     x_mark
     )
```

⇒ `t/nil`

Description

Undoes a transaction back to the last mark, or to start if there are no marks. Supports the Allegro *oops* model for database transactions.

When a transaction mark is committed or rolled back all, then that mark is no longer valid for *oopsing*.

Arguments

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.

axlDBTransactionRollback

```
axlDBTransactionRollback(
     x_mark
     )
```

⇒ `t/nil`

Description

Undo function for a database transaction.

Arguments

x_mark	Database transaction mark returned from `axlDBTransactionStart`.

Value Returned

`t`	Transaction undo completed.
`nil`	Transaction undo not completed.

Example

See `axlDBTransactionStart` for an example.

axlDBTransactionStart

```
axlDBTransactionStart(
     )
```

⇒ *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

none

Value Returned

x_mark	Integer mark indicating transaction start.
nil	Failed to mark transaction start.

Example 1

```
mark = axlDBTransactionStart()
...#1 do stuff ...
axlDBTransactionMark(mark)
...#2 do stuff ...
axlDBTransactionMark(mark)
...#3 do stuff ...

;; do an oops of the last two changes
axlDBTransactionOops( mark ) ; oops out #3
axlDBTransactionOops( mark ) ; oops out #2
axlDBTransactionOops( topList); commit only #1
```

Emulates the Allegro *oops* model.

Example 2

```
i = axlDBTransactionStart()
     ... do stuff ...
j = axlDBTransactionStart()
     ... stuff ...
     axlDBTransactionCommit(j) ;; this is not really commited

j = axlDBTransactionStart()
     ... do more stuff ...
axlDBTransactionRollback(j) ;; oops out "do more stuff"

axlDBTransactionCommit(i) ;; commit changes to database
```

Multiple Start marks.

Note: Database transaction functions do NOT mark select sets. The application handles select set management.

18

Constraint Management Functions

Overview

This chapter describes the AXL-SKILL functions related to constraint management.

For a list of constraints, see Appendix B in the *Allegro Constraint Manager User Guide.*

axlCnsAddVia

```
axlCnsAddVia(
             t_csetName
             t_padstackName
         )
         ==> t/nil
```

Description

Adds padstack to the constraint via list of a physical cset. Padstack does not need to exist to be added to a constraint via list.

If *t_csetName* is `nil`, add padstack to all physical csets.

Note: If a via already exists in the via list, a `t` is returned. Locked csets return a `nil`.

Arguments

t_csetName	Name of physical cset or `nil` for all csets.
t_padstack	Name of a via padstack.

Value Returned

`t`	If added.
`nil`	Error in arguments; cset does not exist or illegal padstack name.

Examples

Add ALLPAD to all csets

```
axlCnsAddVia(nil "ALLPAD")
```

Add ONEPAD to DEFAULT cset

```
axlCnsAddVia("DEFAULT" "ONEPAD")
```

axlCnsAssignPurge

```
axlCnsAssignPurge(
                  s_tableType
       ) ==> x_delCount/ nil
```

Description

Obsolete. Kept for backward compatibility

Purges either the physical or spacing assignment table of unused entries. Allegro PCB Editor supports two assignment tables: physical and spacing. This functionality duplicates that found in the Constraint Assignment Tables forms.

Arguments

`s_tableType`: Spacing or Physical.

Value Returned

`nil` Error.

`x_delCount` Number of entries deleted.

Example

```
axlCnsAssignPurge('spacing)
```

See Also

`axlCnsList`

axlCNSCreate

```
axlCNSCreate(
               g_domain
               t_name
          t_copyName
         )
         ==> t/nil
```

Description

Creates a new electrical constraint set in the domain specified. 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 are created empty for a nil *copyName* . By default, the ECset is created empty. If you provide a second argument, the ECset contents are copied. Electrical csets cannot be created in Allegro PCB Design L.

Arguments

g_domain	Physical, spacing, or electrical domain of cset.
t_name	Name of new cset.
t_copyName	Spacing and physical domains use name DEFAULT electrical, create, and emtpy ECset.

Value Returned

t	Cset created.
nil	Failed for the following reasons: domain name is illegal; name of cset is illegal; cset already exists; or *copyName* cset does not exist.

See Also

axlCNSEcsetCreate , axlCNSDelete , axlCnsList

Example

Create a new physical cset called `foo`.

```
axlCNSCreate('physical "foo" nil)
```

See Also

`axlCNSEcsetCreate`, axlCNSDelete, and `axlCnsList`

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 csets 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 csets in Allegro PCB Design L.

Arguments

g_domain	Physical, spacing, or electrical domain of cset.
t_name	Name of cset.
o_dbidEcset	If an ECset, its *dbid*.

Value Returned

t	Cset deleted.
nil	Cset not deleted because cset does not exist or the cset is locked.

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

t_csetName	Name of physical cset or nil for all csets.
t_padstack	Name of a via padstack.

Value Returned

t	If deleted.
nil	Error in arguments; cset does not exist or illegal padstack name.

Example

Delete via to default cset

```
axlCnsDeleteVia("DEFAULT" "VIA")
```

Delete via to all csets

```
axlCnsDeleteVia(nil "VIA")
```

See Also

axlCnsGetViaList and axlPurgePadstacks

axlCNSDesignModeGet

```
axlCNSDesignModeGet(
     nil

)

⇒ ls_constraints

axlCNSDesignModeGet(
     'all

)

⇒ lls_constraintNModes

axlCNSDesignModeGet(
     'editable

)

⇒ t/nil

axlCNSDesignModeGet(
     s_name/t_name

)

⇒ s_mode/nil
```

Description

Gets the current DRC modes for checks that fall into the set of design constraints. These constraints pertain to the entire board.

To determine the design constraint checks currently supported, use the following command:

```
axlCNSDesignModeGet()
```

Note: Available constraint checks may change from release to release.

Arguments

`nil`	Returns all checks in design type DRC.
`'all`	Returns all checks and current mode.
`'editable`	Returns `t` if mode can be changed, `nil` mode is not changed and when in Allegro PCB Editor studio which does not offer this option.
`s_name`	Symbol name of check.
`t_name`	String name of check.

Value Returned

`ls_names`	List of checks (*`s_name`* ...)
`lls_names`	List of checks and their mode ((*`s_name s_mode`*) ...)
`s_mode`	Mode `'on`, `'off` or `'batch`

Example 1

```
axlCNSDesignModeGet(nil)
```

Gets a current list of design constraints.

Example 2

```
axlCNSDesignModeGet('all)
```

Gets a list of settings for all design constraints.

Example 3

```
axlCNSDesignModeGet('Package_to_Package_Spacing)
```

Gets current setting of package to package.

Example 4

```
axlCNSDesignModeGet("Negative_Plane_Islands")
```

Gets current setting of negative plane islands using a string.

axlCNSDesignModeSet

```
axlCNSDesignModeSet(
     t_name/s_name
     t_mode/s_mode
)
⇒ t/nil

axlCNSDesignModeSet(
     'all
     t_mode/smode
)
⇒ t/nil

axlCNSDesignModeSet(
     l_constraintNModes
     t_mode/smode
)
⇒ t/nil

axlCNSDesignModeSet(
     ll_constraintNModes
)
⇒ t/nil
```

Description

Sets the current DRC modes for design constraints. The modes control the DRC for that design constraint check on the entire board.

To determine the checks that are supported, use the following command:

```
axlCNSDesignModeGet()
```

You can set all checks using the argument `'all`, set individual checks using *t_name*, or set a list of checks to the same mode as follows:

```
'(s_name...) t_mode/s_mode
'(t_name...) t_mode/s_mode
```

You can list sets of checks as follows:

```
'((s_name/t_name s_mode/t_mode)...)
```

For performance reasons, changing modes or values does not invoke DRC. You must manually invoke DRC. You can mark changes in order to perform fewer DRC updates, depending on your changes (see axlCNSMapUpdate on page 949.)

Note: Available constraint checks may change from release to release.

Arguments

s_name	Symbol name of check.
t_name	String name of check.
s_mode	Mode setting may be 'on, 'off, or 'batch.
t_mode	String mode setting may be "on", "off" or "batch"
'all	Returns all checks for a given tier of Allegro PCB Editor.

Value Returned

t	Success
nil	Failure.

Example 1

```
axlCNSDesignModeSet('Package_to_Place_Keepin_Spacing 'on)
```

Turns on package to package keepin check.

Example 2

```
axlCNSDesignModeSet('all 'batch)
```

Makes all design constraints batch only.

Example 3

```
axlCNSDesignModeSet('(Negative_Plane_Islands Pad_Soldermask_Alignment)' off)
```

Turns two constraints off.

Example 4

```
axlCNSDesignModeSet('((Package_to_Place_Keepout_Spacing 'on)) )
```

Sets various constraints to different modes.

For a programming example, see `cns-design.il`, which you can find in the following location:

`<cdsroot>/share/pcb/examples/skill/cmds`

axlCNSDesignValueCheck

```
axlCNSDesignValueCheck(
     s_name/t_name
     g_value
)
⇒ (t_string/nil, nil/t_errorMsg)/nil
```

Description

Checks the syntax of the given value against the allowed syntax for the given constraint. You use the function `axlCNSDesignGetValue(nil)` to get the constraint names.

Note: Allowed syntax may change from release to release.

Arguments

s_name	Symbol name of the constraint.
t_name	String name of the constraint.
g_value	Value to verify

Value Returned

(*t_string*/nil)	Value correct. *t_string* 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/*t_errorMsg*)	Value incorrect. *t_errorMsg* reflects the error.
nil	Arguments are incorrect.

Examples

```
axlCNSDesignValueCheck('Negative_Plane_Islands "10 mils")
```

Tests if allowed to set.

axlCNSDesignValueGet

```
axlCNSDesignValueGet(
     nil
     [g_returnNameString]
)
⇒ ls_constraints

axlCNSDesignValueGet(
     'all
     [g_returnString]
)
⇒ lls_constraintNValues

axlCNSDesignValueGet(
     s_name
     [g_returnString]
)
⇒ f_value/t_value/nil
```

Description

Fetches the values from those design constraints that support values. Use `axlCNSDesignValueGet(nil)` to determine the set of these constraints.

Note: Constraint checks may change from release to release.

Arguments

`nil`	Returns all checks that support values.
`'all`	Returns all checks with values and current value.
`s_name`	Symbol name of value.
`t_name`	String name of value.
`g_returnNameString`	Returns constraint names as strings (default is symbol return.)
`g_returnString`	By default, this returns native type in user units (a float) for all checks supported. If `t`, return is a MKS string where `nil` returns native.

Value Returned

ls_names	List of all controls that support values (symbol.)
lls_constraintNValues	List of all controls with their values '((*s_name f_value/t_value*) ... *f_value* = user unit value, and *t_value* = MKS string value.

Example 1

```
axlCNSDesignValueGet(nil)
```

Gets a list of design constraints that support values.

Example 2

```
axlCNSDesignValueGet('all 't)
```

Gets a list of settings for all design constraints with values returned as MKS strings.

Example 3

```
axlCNSDesignValueGet('Negative_Plane_Islands)
     = 10.0
```

Gets the current setting of `Negative_Plane_Islands` in user units.

Example 4

```
axlCNSDesignValueGet("Pad_Soldermask_Alignment" t)
     = "10 mils"
```

Gets the current setting of `Pad_Soldermask_Alignment` as a MKS string (this passes in inquiry as a string).

axlCNSDesignValueSet

```
axlCNSDesignValueSet(
     t_name/s_name
     f_value/t_value

)

⇒ t/nil

axlCNSDesignValueSet(
     ll_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

s_name	Symbol name of check.
t_name	String name of check.
f_value	Floating point value provided is assumed to be in the default user unit for the constraint. Value may be rounded.
t_value	If given as a string with MKS type, the value is converted to current user units for the constraint. Rounding may result.

Value Returned

`t`	Design constraint value set.
`nil`	Failed to set design constraint value.

Example 1

```
axlCNSDesignValueSet('Negative_Plane_Islands 10.0))
```

Sets a negative plan tolerance to 10 in current database units.

Example 2

```
axlCNSDesignValueSet('Negative_Plane_Islands "10.0 mils")
```

Sets a negative plan tolerance to 10 mils.

Example 3

```
axlCNSDesignValueSet('((Negative_Plane_Islands "20 inches")
     (Pad_Soldermask_to_Pad_Soldermask_Spacing 15.9)))
```

Sets various constraints to different values.

For a programming example, see `cns-design.il` which you can find in the following location:

```
<cdsroot>/share/pcb/examples/skill/cmds
```

axlCNSEcsetCreate

```
axlCNSEcsetCreate(
     t_name
     [t_copyName/o_dbidCopyEcset]
)
⇒ o_dbidEcset/nil
```

Description

Creates a new ECset. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets. The name must be legal and less than the maximum length allowed. Function fails if the ECset already exists.

By default, the ECset is created empty. You can provide a second argument to copy the contents of another ECset into the new ECset.

Arguments

t_name	Name of new ECset.
t_copyName	Optional name to copy from.

Value Returned

o_dbidEcset	*dbid* of the new ECset
nil	Failed due to one of the following: the name is illegal, or the ECset already exists.

Example 1

```
axlCNSEcsetCreate("MyEmptyEcset")
```

Creates a new empty ECset.

Example 2

```
p = car(axlDBGetDesign()->ecsets)
axlCNSEcsetCreate("MyNewEcset" p)
```

Copies the contents of the first ECset in a list.

axlCNSEcsetDelete

```
axlCNSEcsetDelete(
     t_name/o_dbidEcset
)
⇒ t/nil
```

Description

Deletes an ECset from the 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 the ECset is locked, you must unlock it before you can delete it.

Arguments

t_name	ECset name
o_dbidEcset	ECset *dbid*

Value Returned

t	ECset successfully deleted.
nil	ECset is not deleted because of one of the following: the name is incorrect, or ECset is locked.

Example 1

```
axlCNSEcsetDelete("UPREV_DEFAULT")
```

Deletes an ECset by name.

Example 2

```
p = car(axlDBGetDesign()->ecsets)
axlCNSEcsetDelete(p)
```

Deletes the first ECset in a list of ECsets.

axlCNSEcsetGet

```
axlCNSEcsetGet(
     t_name
)
⇒ o_dbidEcset/nil
```

Description

Returns the *dbid* of the ECset when you request it by the ECset name. Electrical Constraint Set (ECset) is a mechanism for grouping a set of electrical constraints and applying them to a set of nets.

Arguments

t_name	ECset name.

Values Returned

o_dbidEcset	*dbid* of the ECset requested.
nil	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
```

Description

Returns the current DRC modes for checks that are members of electrical constraints. These modes pertain to the entire board. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Note: Not all checks are available in all levels of 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.

Value Returned

ls_names	List of checks (*s_name* ...).
lls_names	List of checks and related modes ((*s_name s_mode*) ...)
s_mode	Returns mode 'on, 'off, or 'batch

Example 1

```
axlCNSEcsetModeGet(nil)
```

Lists currently available electrical constraints.

Example 2

```
axlCNSEcsetModeGet('all)
```

Lists settings for all electrical constraints.

Example 3

```
axlCNSEcsetModeGet('Maximum_Stub_Length)
```

Shows current setting of stub length.

Example 4

```
axlCNSEcsetModeGet("Maximum_Via_Count")
```

Shows current setting of via count.

axlCNSEcsetModeSet

```
axlCNSEcsetModeSet(
     t_name/s_name
     t_mode/s_mode

)

⇒ t/nil

axlCNSEcsetModeSet(
     'all
     t_mode/s_mode

)

⇒ t/nil

axlCNSEcsetModeSet(
     l_constraintNModes
     t_mode/s_mode

)

⇒ t/nil

axlCNSEcsetModeSet(
     ll_constraintNModes

)

⇒ t/nil
```

Description

Sets the DRC modes for checks that are members of the electrical constraints set. These modes control the entire board. Electrical Constraint Set (ECset) is a mechanism for packaging up a set of electrical constraints into a group and applying them to a set of nets.

Note: Not all checks are available in all levels of 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 949 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

s_name	Symbol name of the check.
t_name	String name of the check.
s_mode	Mode setting; may be `'on`, `'off`, or `'batch`.
t_mode	String mode setting; may be `"on"`, `"off"`, or `"batch"`.
`'all`	Set all checks for a given tier of Allegro PCB Editor.

Value Returned

`t`	DRC mode set.
`nil`	DRC mode not set.

Example 1

```
axlCNSEcsetModeSet('Maximum_Via_Count 'off)
```

Turns off max via check.

Example 2

```
axlCNSEcsetModeSet('all 'batch)
```

Makes all electrical constraints batch only.

Example 3

```
axlCNSEcsetModeSet('(Maximum_Crosstalk Route_Delay) 'off)
```

Turns two constraints off.

Example 4

```
axlCNSEcsetModeSet( '((Maximum_Crosstalk off)
     (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

s_name	Symbol name of constraint.
t_name	String name of constraint.
g_value	Value to verify.

Value Returned

t	Syntax is correct.
t_errorMsg	Syntax is incorrect. The message indicates the reason.
nil	Constraint name is not supported.

Examples

```
axlCNSEcsetValueCheck('Net_Schedule_Topology "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

o_ecsetDbid	ECset *dbid*.
t_ecsetName	ECset name.
nil	Returns all checks that support values.
'all	Returns all checks with values and current value.
s_name	Symbol name of value.
t_name	String name of value.
g_returnNameString	Returns constraint names as strings (default is symbol return)
g_returnString	Default is to return native type for all checks supported, this is in user units (a float). If t, return is an MKS string where nil returns native.

Value Returned

ls_names	List of all controls that support values (symbol).
lls_constraintNValues	List of all controls with their values as shown: '((*s_name f_value/t_value*) ... *f_value* = user unit value and *t_value* = MKS string value.

Example 1

```
axlCNSEcsetValueGet(nil)
```

Gets a current list of design constraints that support values.

Example 2

```
ecsets = axlDBGetDesign()->ecsets
ecset = car(ecsets)
axlCNSEcsetValueGet(ecset 'all t)
```

Gets a list of settings for all design constraints with values returned as MKS strings.

Example 3

```
axlCNSEcsetValueGet("UPREVED_DEFAULT" 'Maximum_Via_Count)
     = 10.0
```

Gets the current setting of `Maximum_Via_Count` on ECset `UPREVED_DEFAULT`.

Example 4

```
axlCNSEcsetValueGet("UPREVED_DEFAULT" "Pad_Soldermask_Alignment" t)
     = "10 mils"
```

Gets the current setting of `Pad_Soldermask_Alignment` as a MKS string (this passes in inquiry as a string).

axlCNSGetDefaultMinLineWidth

```
axlCNSGetDefaultMinLineWidth(
     t_sublassName
)
=> 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) *`t/nil`*
`allow_ts`	(symbol) NOT_ALLOWED, ANYWHERE, PINS_ONLY, PIN_VIAS_ONLY
`allow_padconnect`	(symbol) ALL_ALLOWED, VIAS_PINS_ONLY, VIAS_VIAS_ONLY, NOT_ALLOWED

`vias`	(string) colon separates the list of via names. Vias are not layer dependent, so are only returned for TOP. Use `axlCnsGetViaList` to get the via list as a list of strings. When `width_max, dp_neck_gap, dp_primary_gap,` and `necklength_max` are set to 0, indicates that this value is not used.

Arguments

`t_cset`	Name of a physical cset. Can use "" for "DEFAULT".
`t_layer`	ETCH layer name (for example, "ETCH/TOP" or "TOP"). If *`nil,`* applies the change to all layers.
`s_constraint`	Name of constraint. If *`nil,`* returns a set of symbol/value pairs of all constraints.
`g_string`	By default, returns value in the native units of the constraint. If *`g_string`* is *`t,`* always returns data as a string.

Value Returned

`g_value`	Value of constraint in design units, except for `same_net`, which is returned as a *`t/nil`*.
`ll_nameValue`	Name values of pairs of physical constraint symbol and constraint value for all physical (*`s_constraint g_value`*).

```
'((necklength_min 10.0) (neckwidth_max 5.0) ...)
```

`ls_cnsTypes`	List of supported physical constraint names.
`nil`	Returns *`nil`* on error (or `allow_etch`).

Example 1

```
axlCNSGetPhysical("" "TOP" 'width_min)
```

Gets the minimum line width in the default cset, TOP layer.

Example 2

```
axlCNSGetPhysical("VOLTAGE" "BOTTOM" nil)
```

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

Documentation same as axlCNSGetSpacing.

Arguments

t_cset:	name of a same net spacing cset. Can use "" for "DEFAULT".
t_layer:	ETCH layer name ("ETCH/TOP" or "TOP"). If `nil` apply change to all layers.

s_constraint: name of constraint. If `nil` returns a set of symbol/value pairs of all constraints.

g_string: By default returns value in the native units of the constraint. If `g_string` is `t`, it will always return data as a string.

Value Returned

g_value: value of constraint in design units

l_nameValue - name value pairs of spacing constraint symbol and constraint value for all spacing. (*s_constraintg_value*). *'((shape_shape 10.0) (line_line 5.0) ...)*

ls_cnsTypes - list of supported same net spacing constraint names.

nil - returns nil 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)
```

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.

axlCNSGetSpacing

```
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

t_cset	Name of a spacing cset. You can use "" for "DEFAULT".
t_layer	ETCH layer name (for example, "ETCH/TOP" or "TOP"). If *nil,* applies change to all layers.
s_constraint	Name of constraint. If *nil,* returns a set of symbol/value pairs of all constraints.
g_string	By default, returns value in the native units of the constraint. If *g_string* is *t*, always returns data as a string.

Value Returned

g_value	Value of constraint in design units, except for same_net, which is returned as *t/nil*.
ll_nameValue	Name value pairs of spacing constraint symbol and constraint value for all spacing (*s_constraint g_value*). '((shape_shape 10.0) (line_line 5.0) ...)
ls_cnsTypes	List of supported spacing constraint names.
nil	Returns *nil* on error (or same_net).

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
```

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 diff 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

nil:	returns all modes that are in spacing domain
all:	returns all checks and current mode
s_name:	symbol name of check.
t_name:	string name of check

Value Returned

ls_names:	list of checks (s_name ...)
lls_names:	list of checks and their mode ((s_name s_mode) ...)
s_mode:	mode 'on, or 'off

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

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 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

s_name:	symbol name of check.
t_name:	string name of check.
s_mode:	mode setting; may be 'on or 'off.
t_mode:	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
```

Description

Documentation same as axlCNSSpacingModeGet.

Arguments

nil:	returns all modes that are in same net spacing domain
'all:	returns all checks and current mode
s_name:	symbol name of check.
t_name:	string name of check

Value Returned

ls_names:	list of checks (`s_name ...`)
lls_names:	list of checks and their mode (`(s_name s_mode)` ...)
s_mode:	mode 'on, or 'off

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

Documentation same as axlCNSSpacingModeSet.

Arguments

s_name:	symbol name of check.
t_name:	string name of check.
s_mode:	mode setting; may be 'on or 'off.
t_mode:	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

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.

Tip

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

t_cset	Cset name. You can use "" for the DEFAULT cset. Use *nil* to apply changes to all csets.
t_layer	ETCH layer name (for example, "ETCH/TOP" or "TOP"). If nil, applies changes to all layers.
s_constraint	Constraint symbol to change. Use `axlCNSGetPhysical(nil nil nil)` for list of permissible values.
g_value	Value to update. For data types, see Data Types above.
ll_constraintValues	Multiple values may be updated by passing a list of lists for the third argument. `'((s_constraint g_value) ... )`

Value Returned

t	Success.
nil	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)
```

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"*.

Tip

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 'same_net nil)
axlCNSSetSpacing("" nil 'line_line 5)
...
axlCNSMapUpdate()
```

Single change calls do not require this.

For a list of current spacing constraints, see axlCNSGetSpacing.

Same_net behavior will change in 16.2. 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 disable same_net checking in all csets, but if you applied a net or constraint area override, it will still be used for those items.

Arguments

t_cset	The cset name. You can use "" for DEFAULT cset. Use *nil* to apply the changes to all csets.
t_layer	The ETCH layer name (e.g "ETCH/TOP" or "TOP"). If *nil,* applies the changes to all layers.
s_constraint	Constraint symbol to change. Use `axlCNSGetPhysical(nil nil nil)` for a list of permissible values.
g_value	Value to update. For data types, see Data Types above.
ll_constraintValues	Multiple values may be updated by passing a list of lists for the third argument. `'((s_constraint g_value) ... )`

Value Returned

t	Success.
nil	An error occurs when the ECsetECset 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

```
axlCNSSetSpacing(nil nil 'same_net nil)
```

Disables `same_net` setting on all csets and all layers.

Example 2

```
axlCNSSetSpacing("" nil 'line_line 5)
```

Sets line-to-line spacing to 5 on DEFAULT cset and all layers

Example 3

```
axlCNSSetSpacing("25_MIL_SPACE" "top" 'same_net "true")
```

Value of DEFAULT cset.

See Also

`axlCNSGetSpacing`, `axlCNSMapClear`, and `axlCNSMapUpdate`

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

t:	no errors
nil:	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

Documentation same as axlCNSSetSpacing.

Arguments

t_cset:	cset name, can use "" for DEFAULT cset. Use `nil` to apply change to all cset.
t_layer:	ETCH layer name ("ETCH/TOP" or "TOP"). If `nil` apply change to all layers.
s_contraint:	Constraint symbol to change. Use `axlCNSGetSameNet`(`nil nil nil`) for list of permissible values.
g_value:	Value to update. For data type, see above for "DATA TYPES".
ll_constraintValues:	Multiple values may be updated by passing a list of lists for the third argument. '((s_contraint g_value) ...)

Value Returned

t	if succeeds
nil	an error

```
- ecset name does not exit
- layer does not exist
- contraint does not exist
- illegal value for constraint
- cset is locked
```

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

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

nil:	returns all modes that are in spacing domain
'all:	returns all checks and current mode
s_name:	symbol name of check.
t_name:	string name of check

Value Returned

ls_names:	list of checks (s_name ...)
lls_names:	list of checks and their mode ((s_name s_mode) ...)

s_mode: 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 be passed as a symbol or a string. For performance reasons, you should either do all yourupdates 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

s_name:	symbol name of check.
t_name:	string name of check.
s_mode:	mode setting; may be 'on or 'off.
t_mode:	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
        ) -> 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.

This requires an XL or better product license.

Arguments

t_layer1	start layer name
t_layer2	end layer name

Value Returned

f_length	via length in design units

Examples

Get length from top to bottom

```
axlViaZLength("TOP" "BOTTOM")
```

See Also

axlCNSGetViaZPVF

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

o_itemDbid	`dbid` of any item that is assigned to a net or Xnet.
t_cnsName	Property name for the constraint to be fetched. This can be either a fixed constraint or a user-defined constraint.
s_name	Symbol name of DRC check (values returned by `axlCNSEcsetModeGet(nil)`. These names may not exactly match the property name. They do not exist for user-defined properties in the ECset.

Value Returned

`t_cnsValue`	Value returned as a string.
`nil`	No value defined for the net.

See Also

axlCnsNetFlattened

Examples:

Net is part of an ECset (electrical constraint set) which has a MAX_EXPOSED_LENGTH constraint:

```
net = 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 949 for a set of interfaces that you use in order to mark changes to perform fewer DRC updates.

Note: Constraint checks may change from release to release.

Arguments

o_ecsetDbid	*dbid* of the ECset.
t_ecsetName	Name of the ECset.
s_name	Symbol name of constraint.
t_name	String name of constraint.

f_value	Floating point value provided is assumed to be in the default user unit for the constraint. Value may be rounded.
t_value	If given as a string with MKS type, the value is converted to current user units for the constraint. Rounding may result.

Value Returned

t	Set value of ECset DRC.
nil	Failed to set value of ECset DRC due to incorrect argument(s).

Examples

Sets impedance:

```
axlCNSEcsetValueSet("UPREVED_DEFAULT"
                    'Impedance ALL:ALL:100.0:2)
```

Sets multi-value:

```
axlCNSEcsetValueSet("UPREVED_DEFAULT"
            '((Impedance "ALL:ALL:100.0:2") (Maximum_Via_Count 5)))
```

axlCnsGetViaList

```
axlCnsGetViaList()
             t_csetName
         )
         ==>lt_padstacks/nil
```

Description

Returns padstacks defined in a physical constraint set. If the cset name is provided then returns only vias assigned for that cset. Otherwise the function returns vias for all csets. The same vias may appear more than once when using the `nil` option. The order of vias has no meaning and returned padstack names may not exist ("*" indicators in cns physical set dialog).

Arguments

`t_csetName`	Name of physical cset.
`nil`	Process all csets.

Value Returned

`lt_padstacks`	List of padstacks defined in a cset or all csets.
`nil`	If no padstacks found or cset not found.

See Also

`axlCnsAddVia`, axlCnsDeleteVia, and `axlCNSGetPhysical`

Examples

Report vias in default physical constraint set

```
axlCnsGetViaList("DEFAULT")
```

Report vias in all physical constraint sets

```
axlCnsGetViaList(nil )
```

axlGetAllViaList

```
axlGetAllViaList (
             [g_attrVias]
         )
         ==> lo_padstack_dbid
```

Description

Returns a list of all padstacks included in via lists in the design. This is a compilation of all via lists from all constraint sets. Optionally it provides padstacks from net VIA_LIST properties.

The order of padstack `dbids` depends on the order of constraint sets, VIA_LIST properties and the associated via lists.

This interface will result in the via padstacks being loaded into the design if they are not already loaded.

Arguments

`[g_attrVias]`	Optional argument to add padstacks that are not included in constraint sets but are provided in some net VIA_LIST attributes.

Value Returned

`lo_padstack_dbid`	List of padstack `dbids`.
`nil`	The design has empty via lists.

axlDRCUpdate

```
axlDRCUpdate (
          g_mode
) -> x_cnt/nil
```

Description

Performs a DRC check on entire design.

Has two return options controled 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.

Caution

Batch mode is being phased out.

Arguments

`g_mode`	`t` do all checks plus batch only checks, `nil` do only interactive checks

Value Returned

`x_cnt`	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

g_mode	t: waive DRC.
	nil: unwaive DRC.
o_DrcDbid	A single DRC marker.
lo_DrcDbid	A list of DRC markers.
t_comment	Optional, add a comment to waived DRC. Only applies in waive mode.

Values Returned

`t`	Success.
`nil`	Failed due to incorrect arguments .

See Also

`axlDBControl`, `axlDRCWaiveGetCount`

Example 1 Waive 1st DRC in drc list

```
p = axlDBGetDesign()->drcs
axlDBGetDesign(t car(p) "This DRC is OK")
```

Example 2 Waive all drcs in design

```
p = axlDBGetDesign()->drcs
axlDBGetDesign(t p)
```

Example 3 Restore all waived DRCs

```
p = axlDBGetDesign()->waived
axlDBGetDesign(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:

nil	Returns DRC error count.
t	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 then the count returned by *g_mode=nil*.

Caution

This is not an efficient way to run batch DRC or "what if" checks.

Arguments

g_mode	*nil*: Returns DRC error count.
	t: Returns list of DRC errors.
o_dbid	A single.

Value Returned

x_cnt	Returns number of errors associated with list of items.
lo_drcDBid	List of DRC *dbids.*

`nil` No *dbids* (if *g_mode* = *t*) 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)
```

axlDRCWaiveGetCount

```
axlDRCWaiveGetCount ()
)⇒ x_count
```

Description

Returns total number of waived DRCs in the design.

Arguments

None.

Value Returned

`x_count` Returns waived DRC count.

axlLayerSet

```
axlLayerSet(
                  o_dbid
        )
        ==>o_dbid/nil
```

Description

Updates changes to layer parameters. You can only update the color and visibility attributes of a parameter. This is a wrapper for `axlSetParam`. After completing color or visibility changes, call `axlVisibleUpdate` to update the display.

Arguments

o_dbid	Layer parameter `dbid`.

Value Returned

o_dbid	Layer parameter `dbid`.
`nil`	If error.

See Also

axlSetParam and `axlLayerGet`

Examples

Change color of top etch layer:

```
q = axlLayerGet("ETCH/TOP")
q->color = 7
q->visibility = nil
axlLayerSet(q)
```

If setting multiple layer colors or visisbility, only call visible update after last change:

```
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

s_csetDomain	Domains supported: spacing, physical, and electrical.
nil	Lists all supported domains.

Values Returned

lt_csetNames	Lists csets in specified domain.
ls_csetDomains	List of supported domains.

See Also

axlCNSCreate

Example 1

```
axlCnsList('spacing)
```

Returns all spacing cset names.

Example 2

```
axlCnsList(nil)
```

Returns supported domains.

axlCNSMapClear

```
axlCNSMapClear(
)
⇒ t
```

Description

See `axlCNSMapUpdate`.

Arguments

none

Value Returned

`t`	Always returns `t`.

Examples

See axlCNSMapUpdate on page 949 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 `axlCNSMapClear`.
`x_drcCount`	Number of DRCs caused by batch changes.

Example 1

```
axlCNSMapClear()
axlCNSEcsetModeSet('Maximum_Via_Count 'off)
axlCNSDesignModeSet('all 'on)
axlCNSDesignValueSet('Negative_Plane_Islands 10.0)
axlCNSMapUpdate()
```

Turns off electrical max via check by turning all design checks on and setting the island tolerance to 10.

Example 2

```
axlCNSMapClear()
axlCNSEcsetModeSet('Maximum_Via_Count 'on)
xlCNSMapUpdate()
```

Does one change.

axlCnsNetFlattened

```
axlCnsNetFlattened(
                  o_netDbid/t_netName
                  t_cnsName/s_name
         )
         ==> t_cnsValue/nil
```

Description

Permits a view of constraints where explicit pinpair rules are promoted to the net. The information reported by the function is the same as in `show element` under the Properties attached to net heading. It is also in a format used by the third party netlist (`netin`) and in the `pstxnet.dat` file used by `netrev`.

If pinpairs are constrained by an electrical rule (for example, PROPAGATION_DELAY), 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 constaint 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 constaints from xnets, matched groups, and pin pairs to nets by promoting or flattening the electrical property to present a traditional net view of the constraints and to provide compatibility with schematic netlisters. Additional constraints may effect the net because of the ECset assigned to the net, xnet, differential pair or bus level. Additional override properties may exist at the differential pair or bus level. You can use `axlNetECsetValueGet`, but it will not flatten constraints.

Tip

When requesting multiple constraints from the same net, use the `dbid` of the net as first argument instead of the net name.

Arguments

`o_netDbid/t_netName` `dbid` or name (string) of the net.

`t_cnsName` Property name for the constraint.

`s_name` Symbol name of DRC check (values returned by `axlCNSEcsetModeGet(nil)`. These names may not exactly match the property name.

Value Returned

t_cnsValue	Value returned as a string exactall.
nil	No value defined for the net.

Examples

Get impedance rule by name on `net1`:

```
rule = axlCnsNetFlattened("NET1" "IMPEDANCE_RULE")
```

Get impedance rule by DRC check name on `net1`:

```
rule = axlCnsNetFlattened("NET1" 'Impedance)
```

Get PROPAGATION_DELAY on MEM_DATA8 using the `dbid` of net:

```
net = car(axlSelectByName("NET" "MEM_DATA8"))
rule = axlCnsNetFlattened(net "PROPAGATION_DELAY")
```

19

Command Control Functions

Overview

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.

axlCmdRegister

```
axlCmdRegister(
     t_allegroCmd
     ts_callback
     ?cmdType t_cmdType
     ?doneCmd ts_doneCmd
     ?cancelCmd ts_cancelCmd
)
⇒ t/nil
```

Description

Registers a command named *t_allegroCmd* with the 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 is hidden.

Once you register a command, Allegro PCB Editor passes its arguments to SKILL-AXL without parsing them.

You can call `axlCmdRegister` any time. Once you have registered a command, you can type it at the Allegro PCB Editor command line and incorporate it into menus and pop-ups.

Tip

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 via 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()`.

Caution

You cannot access interactive Allegro PCB Editor commands via `axlShell` ***if you register your skill code as interactive, because nesting interactive commands are not supported. This includes using*** `axlShell`

to spawn Allegro PCB Editor scripts that contain interactive commands, which are those that display in the lower right side of the Allegro PCB Editor window (where the `Idle` ***string appears).***

Arguments

t_allegroCmd	Name of the command to register. Use lowercase letters for the command name.
ts_callback	Name of SKILL callback routine called when the command is activated from the Allegro PCB Editor window.
t_cmdType	String denoting the type of this command:

`"interactive"`

Allegro PCB Editor interactive command which is the default.

`"general"` 

Immediate command that executes as soon as the command is called, even during another command. Use for display refresh commands, for example.

`"sub_cmd"` 

Must be called inside an interactive command. Use for pop-ups.

ts_doneCmd	*Done* callback function that Allegro PCB Editor calls when the user types `done` or selects *Done* from the pop-up. Can be either a symbol or string. If `nil`, Allegro PCB Editor calls `axlFinishEnterFun` by default.
ts_cancelCmd	Cancel callback function that Allegro PCB Editor calls when the user types `cancel` or selects *Cancel* from the pop-up. This argument can be either a symbol or a string. If it is `nil`, Allegro PCB Editor calls `axlCancelEnterFun` by default.

Value Returned

`t`	Command registered successfully.
`nil`	Command not registered.

Example 1

```
axlCmdRegister( "my swap gates" 'axlMySwapGates
?cmdType "interactive" ?doneCmd 'axlMySwapDone
?cancelCmd 'axlMySwapCancel)
⇒ t
```

Registers the command `my swap gates` as calling the function `axlMySwapGates`.

Example 2 SKILL Function Example

For commands (`s_allegroCmd` value) that accept parameters as strings, the SKILL function converts the parameters to symbols.

```
axlCmdRegister( "do it" 'do_print)

do it myFile Text AshFindAllText
```

Sample `axlCmdRegister` with a registered function that takes arguments where *`myFile`* is an output port.

```
(defun do_print (arg1 description find_func) ;; Added to deal with strings
          myport = evalstring(arg1)            ;; Added to deal with strings
      do_print2( myport description find_func) ;; Added to deal with strings
          ) ; do_print                         ;; Added to deal with strings
# Here is the real function
(defun do_print2 (p description find_func)
     (let (list)
          (fprintf p "Properties on %s:\n" description)
          (setq list (apply find_func nil))
          (print_list_props list p)
          (fprintf p "\n\n")
          ) ; let
) ; do_print2
```

Parse and process the *`do it`* arguments.

axlCmdUnregister

```
axlCmdUnregister(
     t_allegroCmd
)
⇒ t/nil
```

Description

Unregisters or removes from the 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.

Arguments

t_allegroCmd	Name of command to be unregistered.

Value Returned

`t`	Command unregistered successfully.
`nil`	Failed to unregister the specified command.

Example

```
axlCmdUnregister( "my swap gates")
⇒ t
```

Unregisters the command `my swap gates`.

axlEndSkillMode

```
axlEndSkillMode(

)

⇒ t
```

Description

Returns from the SKILL command mode to the program's command line. The SKILL exit function is mapped to this function. In a SKILL program this command has no effect.

Arguments

None

Value Returned

`t`	Always returns `t`.

Example

```
axlEndSkillMode()
⇒ t
```

Exits AXL-SKILL.

axlFlushDisplay

```
axlFlushDisplay(
)
⇒ t
```

Description

Flushes all data from the display buffer to the display screen itself. Displays items intended to be displayed, but not yet displayed because no event has triggered a flush of the display buffer.

You can display the following Items:

- Visible objects added to the database
- Messages
- Highlighting and dehighlighting of selected objects
- Pending display repairs

Generally, AXL delays screen updates until a prompt for user input occurs or an AXL program completes, such as `axlEnterPoint`. Certain programs, such as those that spend long times doing calculations between screen updates, might want to call `axlFlushDisplay` after each batch of screen updates to indicate the progress of the command. Overuse of this call may hurt performance.

Arguments

None

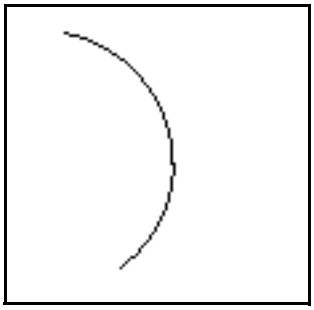
Value Returned

`t`	Always returns `t`.

Example

```
mypath = axlPathStart( list(8900:4400))
axlPathArcRadius(mypath, 12., 8700:5300, nil, nil, 500)
myline = axlDBCreatePath( mypath, "etch/top" nil)
; Arc is not yet visible
axlFlushDisplay()
; Now arc is visible
```

Creates a path and displays it immediately regardless of whether the user has caused a display event such as moving the cursor into the 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

`t`	Suppresses the error message produced when it is not OK to proceed.

Value Returned

`t`	Allegro PCB Editor can allow AXL-SKILL database, selection, and interactive functions to execute.
`nil`	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

g_arcEnable	If `t`, sets Lock Mode to `Arc`. Otherwise Lock Mode is Line. Default is Line.
f_lockAngle	Sets Lock Direction. Allowed values are: 45 (degrees), 90 (degrees), or 0 (off, or no lock).
f_minRadius	Sets Minimum Radius, a value in user units.
f_length45	Sets the Fixed 45 Length value in user units.
g_fixed45	If `t`, sets the Fixed 45 Length mode. You cannot set this parameter unless *f_lockAngle* is 45, and *g_arcEnable* is `nil`.
f_lengthRadius	Sets Fixed Radius value in user units.
g_fixedRadius	If `t`, sets the Fixed Radius mode. You cannot set this parameter unless *f_lockAngle* is 45 or 90, and *g_arcEnable* is `t`.
g_lockTangent	If `t`, sets the Tangent mode to `on`.

Value Returned

`t`	Set the given line lock parameters successfully.
`nil`	Failed to set the given line lock parameters.

Example

```
axlSetLineLock( ?arcEnable t ?lockAngle 90 ?fixedRadius t
?lengthRadius 50)
```

Sets the Line Lock parameters to Lock Direction: `90`, Lock Mode: `Arc`, Fixed Radius: `on` at `30` mils.

axlSetRotateIncrement

```
axlSetRotateIncrement(
     ?angularf_angular
     ?radialf_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

f_angular	Sets angle lock increment in degrees.
f_radial	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:

doneState	How the user returned control to the application. Possible values are either: `done` or `cancel`.
popupId	List of the current pop-up values. A list of string pairs, as shown: `(("Done" "axlFinishEnterFun") ("Cancel" "axlCancelEnterFun"))`
ministatForm	Always `nil`. (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

r_userData	User data structure from Allegro PCB Editor.
`nil`	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

r_popup	Predefined pop-up handle to which to append new entries. Can be `nil` to create a new pop-up.
ts_pairs	List containing the pairs ((*t_display t_callback*) defining each pop-up entry display name and its AXL function callback.

Value Returned

r_popup	Id of the pop-up created or updated.
`nil`	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

r_form	Form handle.
t_field	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

r_form	Form handle
t_field	Field name
t_class	Class name

Value Returned

t	Built form subclass pop-up.
nil	Failed to build form subclass pop-up due to incorrect arguments.

Example

```
     # place holder since popup will be overridden by the code
     POPUP <subclass>"subclass" "subclass".
     ...
     # field name should match t_field
     FIELD subclass
     FLOC 9 3
     ENUMSET 19
     OPTIONS prettyprint ownerdrawn
     POP "subclass"
     ENDFIELD
axlBuildSubclassPopup(fw, "subclass" axlMapClassName("ETCH"))
axlFormSetField(fw, "subclass" "GND")
```

Form file entry provides a subclass pop-up with color swatch support.

Note: `axlMapClassName` supports Allegro Package Designer 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.

Tip

To take advantage of color swatches in your subclass pulldown (ENUMSET) use this interface and the ownerdrawn option in the form file. The form file entry for your control should look like:

FIELD *<field name>*

FLOC *<x y location>*

ENUMSET *<width of field>*

OPTIONS *prettyprint ownerdrawn*

POP *<popup name>*

ENDFIELD

Note *prettyprint* option upper/lower cases the popup name the user sees.

Arguments

r_form	Standard form handle (see axlFormCreate on page 552)
t_field	Field name in form or pop-up name of form.
t_class	Class name.
nil/*lt_subclass*	Use nil 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

`t`	Update now.
`nil`	Update when control is returned to the user.

Value Returned

`t`	Returns `t` 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 visiblity 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.

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.*

20

Polygon Operation Functions

Overview

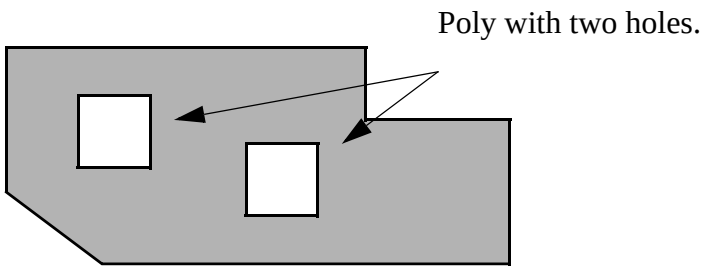
This chapter describes the AXL/SKILL Polygon Operation functions and includes the following sections:

- About Polygon Operations
- AXL-SKILL Polygon Operation Attributes
- AXL-SKILL Polygon Operation Functions
- Use Models

About Polygon Operations

A *poly* is a set of points linked so that the start and end point are the same. A poly is always non-intersecting and may contain *holes*. A hole is a non-intersecting closed loop enclosed within a poly.

Figure 20-1 Poly with Holes



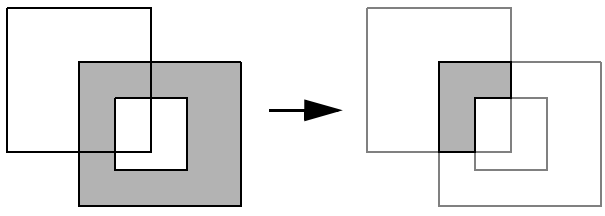
Note: These polys refer to the `o_polygon` object in AXL-SKILL. Polys are different from the AXL Skill Database *polygon* object which represents an Allegro PCB Editor unfilled shape.

These functions, which perform geometric operations on polys, are called logical operations and do the following:

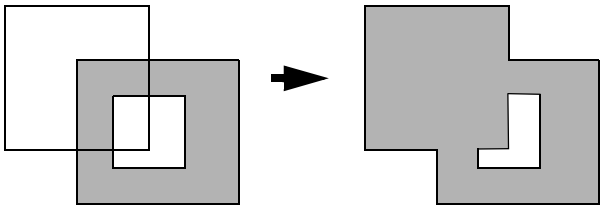
- Create route keepouts based on the board outline, offset by a pre-determined distance.
- Create split planes from route-keepin and Anti-etch.
- Automatically generate a package keepout surrounding a package and its pins, contoured around the package.

Logical operations in AXL-SKILL enable SKILL programmers to do the following:

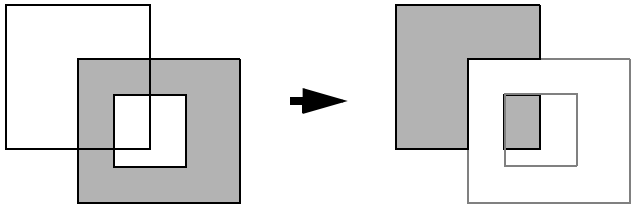
- Create logical operation objects (*lo_polygon*) from these Allegro PCB Editor database objects:
  - pins
  - lines
  - clines
  - vias
  - shapes
  - rectangles
  - frectangles (filled rectangles)
  - voids
- Create *o_polygon* from holes in a poly (*o_polygon*)
- Create an Allegro PCB Editor database 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.



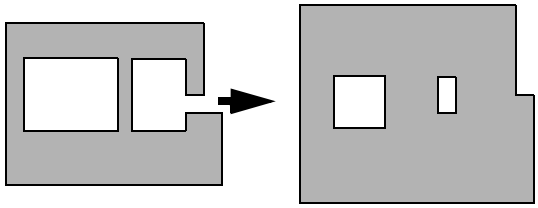
- Logical OR - The union of two polys.



- 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

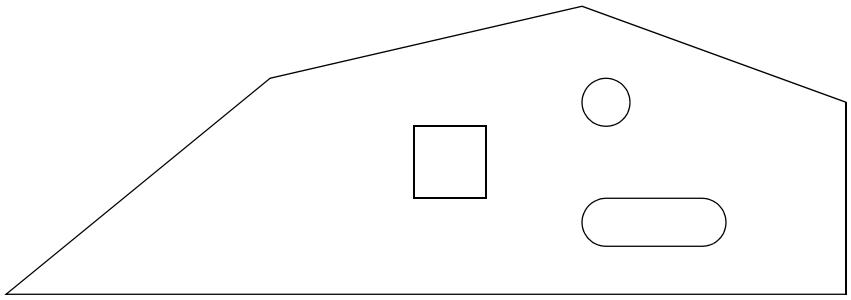
Return values for each function are specified along with the description of the function, in case of error.

AXL-SKILL Polygon Operation Attributes

The following are the attributes of the *o_polygon* type:

Attribute Name	Type	Description
area	float	Area of the poly in the same units as the drawing.
bBox	bBox	Poly's bounding box.
vertices	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.
holes	list	*lo_polygon*
isHole	boolean	`t` = hole, `nil` = poly

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
     ?layert_layer
     ?padType s_padType
     ?holes       t/nil

)

⇒ lo_polygon/nil
```

Description

Creates a list of *o_polygon* objects from the *dbid* or an *r_path*. Use the *lo_polygon* list to get the poly attributes or to perform logical operations on these polys. 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`.

Arguments

o_dbid	axl *dbid* for one of the following: path (line and cline), shape, rect, frect, pin, via, void, arc and line from which to construct the poly. Notes: arc and line are segments reported by `show element`.
r_path	Path construct from the `axlPath` API family. This is not an Allegro PCB Editor database object so it is a much more efficient method than creating an Allegro PCB Editor shape; then converting it to a Poly. Note `axlDBCreateOpenShape` also supports an *r_path*.
s_endCapType	Keyword string specifying the end cap type to use for the poly, one of `'SQUARE`, `'OCTAGON`, or `'ROUND`. Used in case of line or cline only, otherwise ignored. Default is `'SQUARE`.
t_layer	Keyword string specifying the layer of the pad to retrieve, for example, `"ETCH/TOP"`. Used in the case of pin or via only, otherwise ignored. Default is `"ETCH/TOP"`.

s_padType	Keyword string specifying the type of the pad to be retrieved, one of 'REGULAR, 'ANTI, or 'THERMAL. Used in the case of pin or via only, otherwise ignored. Default is 'REGULAR.
holes	Default value is t. By default, for shapes with voids returns any voids as holes. If the value is set to nil, does not return the holes.

Value Returned

lo_polygon	Object representing the resulting geometry.
nil	Cannot get polys.

Creates an *lo_polygon* object from a database object. 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

See documentation for individual use.

Example 1

Create a poly from a via

```
polyList = axlPolyFromDB(via_dbid, ?layer "ETCH/BOTTOM" ?padType 'ANTI)
```

Example 2

Create a rectangle poly (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->??
```

axlPolyMemUse

```
axlPolyMemUse (
) -> lx_polyCounts
```

Description

This returns a list of integers reflecting the internal memory use of the `axlPoly` interfaces. If you assign Poly objects to global handles (instead of assigning to locals, e.g `let` or `prog` statements) then you need to insure all of global data is `nil`-ed at the end of your program. The example below shows how to check that you have written your program correctly.

Description of 5 integers. Integers 2 through 5 are for Cadence use.

1 - Most important and shows number of Skill Polys still in use.

2 - Number of Allegro Polys in use. This is always >= to Skill Polys. The additional polys are voids (holes) in the Skill polys.

3 - Number of edges in all Allegro polys.

4 - Number of Allegro Floating Point Polys (should be 0).

5 - Number of edges in all Allegro Floating Point Polys (should be 0).

Arguments

None

Value Returned

`lx_polyCounts`	A list of 5 integers reflecting Poly memory usage.

See Also

`axlPolyOperation`

Example

Verify at end of your program you have no hanging Poly memory in use.

```
gc() ; requires Skill development licenses
```

```
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	Coordiates in user units for offset.
g_copy	Otional, if `t` does the offset on a copy.

Value Returned

lo_polygon/

o_polygon	In place offset (`g_copy nil`) or offseted copy of polygond (`g_copy` is `t`). If passed a list of polys returns a list otherwise return a poly.

See Also

axlPolyFromDB

Example

See the following.

```
<cdsroot>/share/pcb/examples/skill/axlcore/ashpoly.il
```

axlPolyOperation

```
axlPolyOperation
     o_polygon1 / lo_polygon1
     o_polygon2 / lo_polygon2

s_operation

)

⇒ lo_polygon/nil
```

Description

Performs the logical operation specified on the two sets of polys. Does not allow hole polys as input. When holes are passed as input, the following warning is displayed:

```
Invalid polygon id argument -<argument>
```

Underlying polygon operation function fails and returns `nil` in rare dense geometrical situations.

Arguments

o_polygon1 / *lo_polygon1*	*o_polygon* or list of *o_polygons* on which the operation is to be done.
o_polygon2 / *lo_polygon2*	List of *o_polygons* on which the operation is to be done.
s_operation	String specifying the type of logical operation, one of `'AND`, `'OR`, or `'ANDNOT`.

Value Returned

lo_polygon	List of *o_polygons* which represent the geometry resulting from performing the operation on the arguments.
`nil`	Error due to incorrect arguments.

To be more specific:

(*o_polygon_out1 o_polygon_out2* ...) is returned if the result after performing the operation is a list of polys.

`nil` is returned if the result after performing the operation is a `nil` poly. For example, consider performing the `AND` operation on two non-overlapping sets of polys.

`nil` is returned if the operation fails. You can obtain a descriptive error message by calling `axlPolyErrorGet()`.

Example

```
poly1_list = (axlPolyFromDB cline dbid)
     poly2_list = (axlPolyFromDB shape_dbid)
     res_list = (axlPolyOperation poly1_list poly2_list 'OR)
```

axlPolyExpand

```
axlPolyExpand(
     o_polygon1 / lo_polygon1
     f_expandValue
     s_expandType
     )
```

⇒ *lo_polygon*/nil

Description

This function yields a list of polys after expanding them by a specified distance. Use of a negative number causes contraction. Distance is specified in user units. This function does not allow hole polys as input. When holes are passed as input, the following warning is displayed:

```
Invalid polygon id argument -<argument>
```

Caution

Underlying polygon operation function fails and returns **`nil`** ***in rare dense geometrical situations.***

Arguments

o_polygon1 /lo_polygon1

o_polygon / list of *o_polygons* on which the operation is to be done.

f_expandValue	Amount of expansion in user units.
s_expandType	Symbol specifying the exterior corners of the geometry during expansion, one of:

```
'NONE          - no corner modifications
'ALL_ARC
'ACU_BLUNT
'ACU_ARC
```

Value Returned

lo_polygon	List of *o_polygons* which represent the resulting geometry after performing the expansion on the specified polys.
nil	Failed to expand polys due to incorrect arguments.

To be more specific:

(*o_polygon_out1 o_polygon_out2* ...) is returned if the result after performing the operation is a list of polys.

`nil` is returned if the result after performing the operation is a `nil` poly, for example, consider contracting a 20x30 rectangle by 40 units.

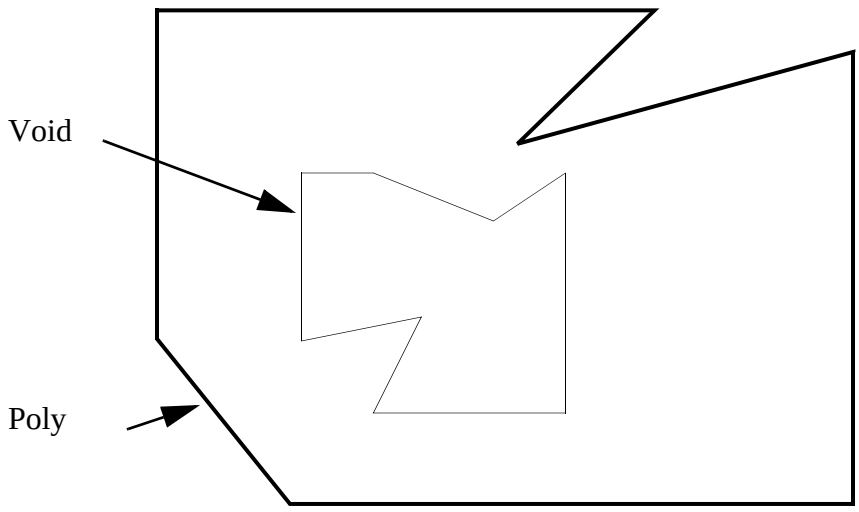
`nil` is returned if the operation fails. You can get a descriptive error message by calling `axlPolyErrorGet()`.

Example

```
poly_list = (axlPolyFromDB shape_dbid)
     exp_poly = (axlPolyExpand poly_list 10.0 'ALL_ARC)
```

The following sequence of diagrams illustrates the behavior of each of the options.

Figure 20-2 Original Poly

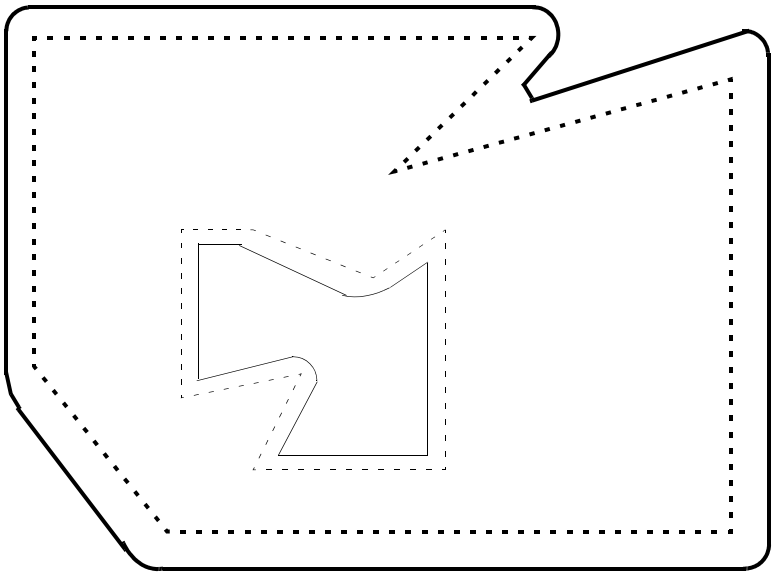


ALL_ARC

During expansion of the poly boundary, an arc is inserted for the edges in the offset shape that satisfy the following criteria:

- Edges form an *outside* (or convex) point of the poly boundary. The reverse is true for the voids.

Figure 20-3 Expanded Using ALL_ARC



Legend:

Original Polygon

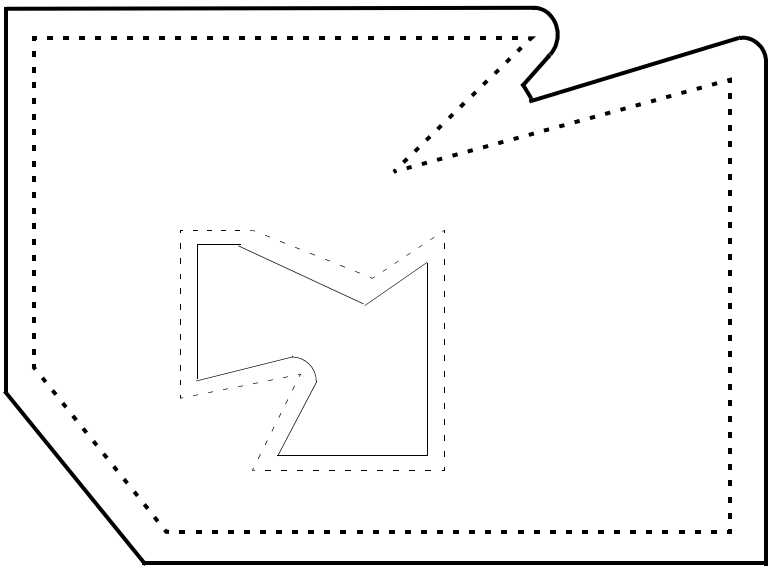
Expanded Polygon

ACU_ARC

During expansion of the poly boundary, an arc is inserted for the edges in the offset shape that satisfy the following criteria:

- Edges form an *outside* (or convex) point of the poly boundary.
- Edges form an angle sharper than 90 degree.
  The reverse is true for the voids.

Figure 20-4 Expanded Using ACU_ARC



Legend:

Original Polygon

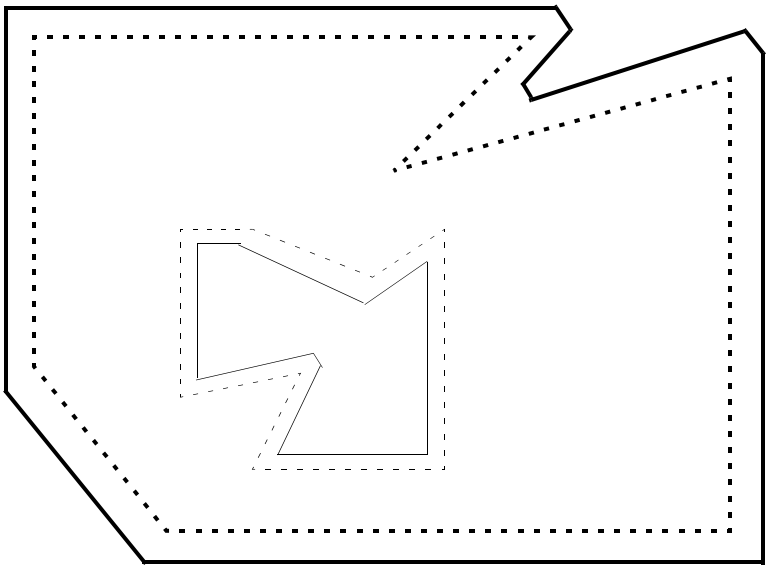
Expanded Polygon

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 20-5 Expanded Using ACU_BLUNT



Legend:

Original Polygon

Expanded Polygon

axlIsPolyType

```
axlIsPolyType (
     g_polygon
)
⇒ t/nil
```

Description

Tests if argument *g_polygon* is a polygon user type.

Arguments

g_polygon	Object to test.

Value Returned

t	*g_polygon* is a polygon user type.
nil	*g_polygon* is not a polygon user type.

Example

```
poly = axlPolyFromDB(cline_dbid)
axlIsPolyType(poly) returns t.
axlIsPolyType(cline_dbid) returns nil.
```

axlPolyFromHole

```
axlPolyFromHole (
     o_polygon
)
⇒ lo_polygon/nil
```

Description

Creates a new poly from the vertices of the hole, and sets the *isHole* attribute of the resulting poly to `nil`. Function returns `nil` in case of error.

Arguments

o_polygon	*o_polygon* on which the operation is to be done. Must have *isHole* attribute set to `t` (that is, the argument must be a hole).

Value Returned

lo_polygon	List of *o_polygons* which represent the resulting geometry after creating poly from the hole argument.
`nil`	Error due to incorrect argument.

Example

```
poly = axlPolyFromDB(shape_dbid)
hole = car(poly->holes)
polyList = axlPolyFromHole(hole)
```

axlPolyErrorGet

```
axlPolyErrorGet (

)

⇒ t_error/nil
```

Description

Retrieves the error from the logop core. See the following list of error strings returned by the logical operation core:

Error type	String returned
problem with arcs	"Bad arc data in polygon operations."
bad data	"Data problem inside polygon operations."
internal error in logical op data handling	"Polygon operation failed because of internal error."
numerical problem in logical op	"Computational problem while doing polygon operations."
memory problem	"Out of memory."
no error	NIL

Arguments

None.

Value Returned

t_error	Error from the logical operation core.
nil	No logical operation error.

Example

```
l_poly1 = axlPolyFromDB(shape_dbid)
l_poly2 = axlPolyFromDB(cline_dbid ?endCapType 'SQUARE)
l_polyresult = axlPolyOperation(l_poly1 l_poly2 'ANDNOT)
if (null l_polyresult) axlMsgPut(list axlPolyErrorGet())
```

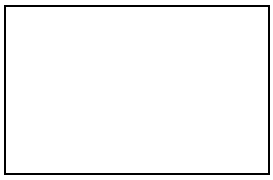
Use Models

Example 1

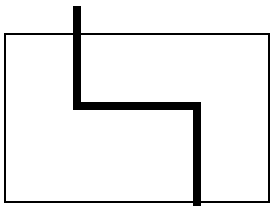
The existing Split Plane functionality can use the AXL version of the logical operation.

The objective is to split the route-keepin shape on the basis of the anti-etch information and add the resulting split-shapes to the database.

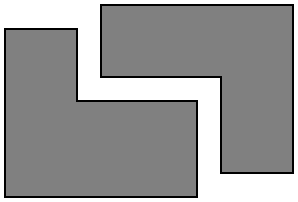
Split Plane Usage



1. startShape - route keepin rectangle



2. antiEtchGeom - anti-etch line on layer “ANTI ETCH/XYZ”



3. The above two shapes are created after doing the operation ANDNOT and then creating the shape for the resultant polys.

```
; retrieve the route keepin rectangle
startShape = (myGetRouteKeepin)
; retrieve the shapes and lines on the anti-etch subclass
antiEtchGeom = (myGetAntiEtchGeom (axlGetActiveLayer))

; create the polygon for the route-keepin rectangle
startPoly = (axlPolyFromDB startShape)
; create the polygon for all the anti-etch elements
antiEtchPoly = nil
(foreach antiElem antiEtchGeom
     antiElmPoly = (axlPolyFromDB antiElem ?endCapType 'ALL_ARC)
     antiEtchPoly = (append antiElmPoly antiEtchPoly)
)
; do the LogicalOp operation
splitPolyList = (axlPolyOperation startPoly antiEtchPoly 'ANDNOT)

;check for any error in logop
(if splitPolyList then
     (; add the resultant polygons as a set of filled shapes on the
     ; active class/subclass with no net name.
     (foreach resPoly splitPolyList
          (axlDBCreateShape resPoly t)
     ))
```

```
else
     (axlMsgPut(list axlPolyErrorGet()))
)
```

Example 2

```
; retrieve the polygon corresponding to clock gen
clockPoly = (axlPolyFromDB rect_id)
; get polygon associated with the ground shape
gndPoly = (axlPolyFromDB shp_dbid)
; get the intersection of the two polygons
shieldedPoly = (axlPolyOperation clockPoly gndPoly 'AND)

; check for any error in logop
(if shieldedPoly then
     (; get the area of the intersection polygon
     shieldedArea = shieldedPoly->area
     ; get the area associated with clock gen
     clockArea = clockPoly->area

     ;get the ratio of the two areas
     ratioArea = shieldedArea / clockArea)
else
     (axlMsgPut(list axlPolyErrorGet()))
)
```

Gets the shielded area of a clock generator by a ground shape using the rule stating that the ratio of the shielded area to the actual area should be less than a particular threshold.

21

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.

Important

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

`nil`	No error message available.
`t_errorMessage`	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

t_id	Id describing file attributes from `fileops.txt`
t_name	Name of file to find.
t_mode	Open mode. One of the following:
	`r:` read-only
	`w:` write
	`wf:` write line-buffered
t_prop	Property string.

Value Returned

t_name	Name of file opened.
`nil`	Failed to open file.

See Also

axlDMOpenFile

Example

```
(setq aPort(axlDMFindFile "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

t_id	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`.
t_name	String giving name of the file to open.
t_mode	Open mode. One of the following: `r`: read-only `w`: write `wf`: write line-buffered
t_prop	Property string.

Value Returned

t_name	Name of the file. In the case of *t_mode* = "r", the file must exist for successful completion.
nil	File not found. Displays a confirmer giving the name of the file it could not find.

See Also

axlDMOpenFile

Example

```
myfile = axlDMGetFile( "ALLEGRO_TEXT" "clip" "r")
     ⇒ "/usr/home/fred/myproj/clip.txt"
```

Finds the file clip.txt, available for reading.

axlDMOpenFile

```
axlDMOpenFile(
     t_id
     t_name
     t_mode
)
⇒ p_port/nil
```

Description

Opens a file in 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.

Tip

Use `get_filename(p_port)` to obtain the name of the file.

Arguments

t_id	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`.
t_name	String giving the name of the file to open.
t_mode	Open mode. One of the following:

`r`: read-only

`w`: write, create if doesn't exist, truncate to zero length if exists

`a`: open for writing, create if doesn't exist, go to end of file for appending if exists

In addition, the following modifiers are supported

`f`: Flush file after each write. This can be slow on Windows if writing across the network. This is typically used if a process will take a long time and 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

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 confirmer giving the name of the file it could not find.

See Also

axlDMFileError, axlDMFindFile, axlDMGetFile, axlDMOpenLog, axlDMClose, axlDMFileParts

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")
```

axlDMOpenLog

```
axlDMOpenLog(
     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 confirmer giving the name of the file it could not find.

See Also

axlDMOpenFile

Example

```
logport = axlDMOpenLog("clipboard"))
```

Opens the file `clipboard.log` for writing.

axlDMClose

```
axlDMClose(
     p_port
)
⇒ t/nil
```

Description

Closes an opened Allegro PCB Editor file. While you may close a file via the core SKILL function, `close`. It is suggested that any file opened via an `axlDM` function be closed via this function. It your program adheres to this standard then it will be compatible with future Allegro Data Management enhancements.

Use this in place of Skill's infile/outfile if you have used `axlDMOpenFile` or `axlDMOpenLog`.

Arguments

p_port	Id of the open port to be closed.

Value Returned

`t`	Closed the file.
`nil`	File not found.

Example

```
mylog = axlDMOpenLog("myapplic")
     ⇒ port: "/usr/home/fred/myproj/myapplic.log"
axlDMClose(mylog)
     ⇒ t
```

Opens and closes the file `myapplic.log`.

axlDMBrowsePath

```
axlDMBrowsePath(
     t_adsFileType
     [t_title]
     [t_helpTag]

)

⇒ t_fileName/nil
```

Description

Invokes a standard 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

t_adsFileType	First entry in `fileops.txt`.
t_title	Title for the dialog
t_helpTag	Tag for the help file (used only by Cadence)

Value Returned

t_filename	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

t_startingDirectory	Name of the starting directory.
g_writeFlag	A boolean - if the file is to be opened for write (`t`), or for read (`nil`).
t_helpTag	Defines the help message to display if the *Help* button is selected in the browser. Default help is provided if this option is not set.
g_title	Override default title bar of the browser. Normally, this is the name of the command that invoked the browser.

Value Returned

t_dirName	Name of directory selected.
`nil`	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

t_id	Id describing the file attributes from `fileops.txt`, or list of ids for different types, or `nil` if you use `optFilters` to describe files.
g_writeFlag	If the file is to be opened for write (`t`), for read (`nil`).
t_defaultName	Name of file to select by default.
t_helpTag	Tag that defines the help message to display if the Help button is selected in the browser. Default help provided if option not set.
g_directorySet	Sets the directory change button which, by default, is not set.
g_noDirectoryButton	Hiding of the directory change button in the browser. By default, the button is present.

g_noSticky	File browser normally remembers the directory from the previous invocation. This helps the user who browses in the same location that is different from the current working directory. If `t`, then it starts the browser in the current working directory. Normally, you should set this option if *g_directorySet* is `t`.
g_mainFile	Matches options Allegro PCB Editor uses to open files from the File menu. This is *g_noSticky*=`t` & *g_directorySet*=`t`. For non-main files, use no options.
g_title	Overrides default title bar of the browser. Normally this is the name of the command that invoked the browser.
g_filters	Filters added to default `t_id` filter. The format is: <*msg*>|<*filter*>|<*msg*>|<*filter*>...

Value Returned

t_fileName	Name of the file selected.
`nil`	No file selected.

Example 1

```
axlDMFileBrowse("ALLEGRO_TEXT" nil)
```

Browses Allegro PCB Editor text files.

Example 2

```
axlDMFileBrowse("ALLEGRO_TEXT" nil ?optFilters "All log files|*.log|")
```

Browses Allegro PCB Editor text files and allows secondary filter of `*.log`.

Example 3

```
axlDMFileBrowse(nil nil ?optFilters "Skill files(*.il)|*.il|")
```

Browses SKILL files.

axlDMFileParts

```
axlDMFileParts(
     t_filespec
     )
```

⇒ (*directory file fileWext ext*)

Description

Breaks a filename into it's component parts.

Arguments

t_filespec	Filename or full path spec.

Value Returned

list	(*directory file fileWext ext*)

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

t_src	Full path of the source file.
t_dest	Full path of the destination file.
g_append	Flag for the append function (t/nil)

Value Returned

t	Copied file.
nil	Failed to copy file due to incorrect arguments.

Example

```
unless(axlOSFileCopy("~/myfile" "~/newfile" nil)
     axlUIConfirm("file copy FAILED") )
```

axlOSFileMove

```
axlOSFileMove(
     t_src
     t__dest
     )
```

⇒ t/nil

Description

Moves the given source file to the given destination.

Arguments

t_src	Full path of the source file.
t_dest	Full path of the destination file.

Value Returned

t	Moved file.
nil	Failed to move file.

Example

```
unless (axlOSFileMove("/mydir/myfile" "/newdir/newfile")
     axlUIConfirm("file move FAILED") )
```

axlOSSlash

```
axlOSSlash(
     t_directory
     )
```

⇒ *t_directory*/nil

Description

Changes DOS style backslashes to UNIX style slashes which are more amenable to SKILL. On UNIX, returns the incoming string.

Arguments

t_directory	Given directory path.

Value Returned

t_directory	Directory path using UNIX style slashes (/).
nil	Failed due to incorrect argument.

Example

```
p = axlOSSlash("\tmp\mydir")
     -> "/tmp/mydir"
```

axlRecursiveDelete

```
axlRecursiveDelete(
     t_directory
     )

⇒ t/nil
```

Description

Recursively removes directories and subdirectories in the argument list. Directory is emptied of files and removed. If the removal of a non-empty, write-protected directory is attempted, the utility fails. If it encounters protected files or sub-directories, it does not remove them or the parent directories, but removes all other objects.

This can be dangerous since it can severely damage your system or data if not used with care. For example, axlRecursiveDelete("/") could delete your OS and all of your data.

Arguments

t_directory	The given directory or filename.

Value Returned

t	Directory is successfully removed.
nil	Failed for one of the following reasons: - doesn't exist - read protected - sub-file or directory does not allow remove (*partial success*) - sub-file or directory is in use (NT only) (partial success) A partial success means that some of the files and directories were deleted.

Example

```
parent = "./tmp"
child = (strcat parent "/child")
(createDir parent)
(createDir child)
(axlOSFileCopy"~/.cshrc" (strcat parent"/csh") nil)
(axlOSFileCopy"~/.cshrc" (strcat child"/csh") nil)
(axlRecursiveDelete parent)
```

axlTempDirectory

```
axlTempDirectory(
     )
⇒ t_directoryName/nil
```

Description

Returns the temporary directory for the current platform.

Arguments

none

Value Returned

t_directory	Temporary directory for the current platform.
nil	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

g_local	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

t_tempFileName	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.

22

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

t_viewFile	Name of the extract command file (view file).
lt_resultFiles	String or list of strings giving the names of the files to which to write the extracted ASCII data. If you designate only one output file, this is a string.
lt_options	List of keyword strings for various options:
"crosshatch"	Generate crosshatch lines when extracting crosshatched shapes.
"ok"	Unknown field names are OK. Useful when the extract command file has property names not defined for the design being extracted.
"short"	Extract data in the short format
"quiet"	Do not print progress messages on `stdout`.
"mcm"	Output MCM terminology.

Value Returned

t	Extract complete.
nil	Failed to complete the extract. See the file `extract.log` 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

t_viewFile	Name of the extract command file (view file).
s_applyFunc	SKILL function to call for each selected object.
g_userData	Data to pass to *s_applyFunc*. May be nil. Ignores *g_userData* if *s_applyFunc* is nil.

Value Returned

t	*s_applyFunc* is non nil. Applied SKILL functions to specified objects.
l_dbid	List of *dbids* 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
```

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
```

Description

Allows registration of user reports using the Allegro PCB Editor report dialog box. You provide a report name, title, and a report-generating callback function. Callback function format is: `g_reportCallback(t_outfile)`.

If you generate the report, returns a `t` from the function, or `nil`, if you do not generate a report. If you provide a `nil` *`t_description`* then the current report is unregistered. You can disregard the *`t_title`* in the unregister mode. You can register only one report per callback function (`g_reportCallback`). To delete an existing report, pass the callback function assigned to the report, a `nil` for the *`t_description`*, `abd` an empty string for the *`_title`*.

Note: Only applicable if you enable the *HTML-style reports*.

You can generate the report file in two formats: text or HTML.

Text:

- File is text based.
- Conversions include inserting links when following keywords are encountered:

  (http://) - add an HTML link

  (num <,> num) - XY coordinate; add a cross-probe link to zoom center on that coordinate

HTML:

- File starts with an <HTML>
- To enable xy zoom center, decorate all xy coordinates as follows:

  ```
  <a href="xprobe:xy:<x>,<y>">(x y)</a>
  ```

  Example: xy coordinate is (310 140)

  ```
  <a href="xprobe:xy:310.0,140.0">(310.0 140.0)</a>
  ```

Suggestions and restrictions:

- Your report description string should start with your company name or some other standard to keep all your reports together in the report dialog. Keep the description short to fit within the space provided in the dialog box.
- The report dialog is blocking. Do not use the following:

  APIs:

  - any user pick or selection API, `axlSelect`, `axlEnter`.
  - `axlShell`
  - save or open a drawing.
- You can invoke your own form within the report callback but make sure it is blocking (use `axlUIWBlock(<formhandle>)`)
- Return t if you want to the report to display. A nil will not display the report
- If building a context where your report function is stored, you cannot make the `axlReportRegister` inside the context.The `axlReportRegister` must be placed in `allegro.ilinit` or the `.ini` file (if building autoload contexts). This is the same rule that applies to `axlCmdRegister`.

Arguments

`g_reportCallback`	Symbol or string of a SKILL function.
`t_name`	Name of report (shown in reports dialog box); if `nil` will delete the report associated with `g_reportCallback`.
`t_title`	Name in title bar when displaying report.

Value Returned

`t`	Arguments correct.
`nil`	Illegal argument.

Examples

The following registers the report My Hello. The optional keyword in MyReport lets a direct call to MyReport generate a report to a fixed name file, `helloWorld.rpt`. otherwise it takes the name from the report dialog box.

```
axlReportRegister('MyReport, "XYZ Inc. Hello" "Hello World")

             ; optional keyword allows a 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

23

Utility Functions

Overview

The chapter describes the AXL-SKILL utility functions. These include functions to derive arc center angle and radius, and to convert numeric values to different units.

axlCmdList

```
axlCmdList(

)
⇒ ll_strings/nil
```

Description

Lists commands registered by `axlCmdRegister`. The list consists of paired strings.

((<*allegro command*> <*skill function*>)...)

Arguments

None.

Value Returned

ll_strings	List of registered Allegro PCB Editor commands paired with the related SKILL functions.
`nil`	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

`t`	To enable AXL Skill debug mode.
`nil`	To disable.

Values Returned

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 (`f_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

t_filename	the name of the plot file which contains the detail.
point	the location to place the detail.
f_scale	the scaling factor of the details as a floating point number (1.0 is no scaling). Most be greater then 0.1
x_rotation	the rotating angle in degrees. Rotation is restricted to integers.
g_mirror	whether the detail should be mirrored (t/nil.

Value Returned

t	was able to load
nil	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"
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

axlDetailSave

```
axlDetailSave(
                t_filename
                l_bBox
                o_dbid/lo_dbid
        )
        ==> 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.
- Ignoreds logical dbids such as nets, components, etc.

Arguments

t_filename	the name of the file into which the detail is saved
l_bBox	list of two xy coordinates defining selection box
lo_dbid	list of AXL DBID's or single DBID

Value Returned

t	was able to save into the file
nil	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 confirmer 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

t_to	E-mail address.
t_cc	Any carbon copy persons to send; `nil` if none.
t_subject	What to put on the subject line.
t_body	Body of message.

Value Returned

`t`	If successful.
`nil`	Failure.

Example

```
axlEmail("aperson" nil "A test message" "anybody home?")
```

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.

Value Returned

`t`	URL displayed in web browser.
`nil`	Failure due to web browser not being found (on UNIX), not being registered (on Windows), or being unable to load the URL.

Note: The function may not detect when a URL has not loaded on UNIX.

Examples

```
axlHttp("http://www.cadence.com")
```

axlIsDebug

```
axlIsDebug(

) ==> t/nil
```

Description

This checks if AXL debug mode is enabled. This is associated with the `axldebug` Allegro environment variable.

When this mode is set, AXL may print additional AXL API argument warnings when arguments to AXL functions are incorrect. Many warnings do not obey this variable because they are considered serious or edge conditions. Warnings supported are less serious and are known from user feedback to be troublesome to program around.

Typically, when an AXL API function encounters an argument error, it will print a warning and return `nil`.

When unset, the code runs in development mode and eliminates many of these warnings. You should have this mode enabled when developing Skill code.

This functionality was introduced in 15.2. Currently, few AXL functions take advantage of this option.

Arguments

None

Value Returned

`t` if mode is enabled, `nil` otherwise.

See Also

`axlDebug`

Example

```
when( axlIsDebug() printf("Error\n"))
```

axlIsProductLineActive

```
axlIsProductLineActive(t_productLine) -> t/nil
```

Description

This routine determines if a product in a given product line has been been started.

Arguments

`productLine` The product line that you want to check.
The legal values are:

- SI
- PCB
- CONCEPT
- ORCAD
- APD
- SIP
- PACKAGING

Note: PACKAGING" is equivalent to "APD" + "SIP"

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.

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

p_port	SKILL port for the open file.
t_titleString	Title string in the log file header.
t_prefix	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.

axlMKS2UU

```
axlMKS2UU(
     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.

Example 1

```
axlMKS2UU("100.1") -> 100.0
```

Default conversion with database in mils.

Example 2

```
axlMKS2UU("100 mils") -> 100.0
```

Database is in mils.

Example 3

```
axlMKS2UU("100 inches") -> 100000.0
```

Database is in mils.

Example 4

```
axlMKS2UU("100 METER") -> 3937008.0
```

Database is in mils.

axlMKSAlias

```
axlMKSalias(
     t_MKSAlias
)
⇒ t_alias/nil
```

Description

Searches the MKS unit database for the current definition associated with *unitName*. Unlike axlMKSConvert, this function does not convert.

You load the MKS database from the following file:

```
<cds_root>/share/pcb/text/units.dat
```

Argument

t_mksAlias	Name of the alias string.

Value Returned

t_def	Definition name as a string.
nil	Definition name could not be found.

Examples

```
axlMKSAlias("VOLTAGE") -> "V" (Intended usage)
axlMKSAliaas("M") -> "METER"
axlMKSAlias("KG") -> NIL (The function supports only the use of basic units.)
```

axlMKSConvert

```
axlMKSConvert(
     n_input
     t_inUnits
     [t_outUnits]

)

⇒ f_output/nil
```

Description

Converts any allowable units to any other allowable units. Operates in several ways, depending on the arguments.

Note: The synopsis above shows only one of several possible argument combinations. The descriptions below detail each combination.

In all cases where this function actually does a conversion, returns the converted value as *f_output*. If it cannot convert the number for any reason, it returns `nil`. The one set of arguments to `axlMKSConvert` that allow `nil` is the input units pre-register call.

Arguments 1

```
axlMKSConvert(
     n_input
     t_inUnits
     [t_outUnits]

)
      -> f_output/nil
```

Converts the number *n_input* from *t_inUnits* to *t_outUnits* and returns it as *f_output*.

n_input	Number to convert
t_inUnits	String giving the units of the input number
t_outUnits	String giving the new units for `f_output`. If *t_outUnits* is `nil`, converts the number to the current units of the layout.

Value Returned 1

f_output	Converted value of *n_input.*

`nil`	Failed to convert value due to incorrect arguments.

Example 1

```
(axlMKSConvert .5 "MILS" "INCHES) -> 0.0005
```

Arguments 2

```
axlMKSConvert(
     t_input
     [t_outUnits]

)

⇒ f_output/nil
```

Converts the string specifying a value and its units *f_output* to *t_outUnits* and returns it as *f_output*.

t_inUnits	String giving the input number to convert and its units
t_outUnits	String giving the new units for f_output. If *t_outUnits* is nil, converts the number to the current units of the layout.

Value Returned 2

f_output	Converted value of *n_input.*
`nil`	Failed to convert *n_input* due to incorrect argument(s).

Example 2

```
(axlMKSConvert ".5 MILS" "INCHES) -> 0.0005
```

axlMKSStr2UU

```
axlMKSStr2UU(
     t_String
)
⇒ t_mksString/nil
```

Description

Converts an input string to a MKS string in current database units. If the input string is in MKS units, that is used as the basis for the conversion. If the string has no units, the function uses the default database units for the string.

Conversion may fail for the following reasons:

- Input string is not a legal MKS format.
- Conversion overflows the maximum database size allowed.

Notes:

- Conversion between metric and english units may result in rounding.
- Number returned is rounded to the database precision.

Argument

t_String	Input string.

Value Returned

t_mksString	MKS string in current database units.
nil	Failed to convert input string. See earlier Description for possible reasons.

Example

```
axlMKSStr2UU("100.1") -> "100.0 MILS"
```

Default conversion with the database in mils.

axlMapClassName

```
axlMapClassName(
     t_oldName
     [g_mapToPCB]
)
⇒ 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 23-1 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

Argument

t_oldName	Allegro PCB Editor class name.
g_mapToPCB	Optional. `t` maps from the APD name to the PCB names. Default is `nil` (PCB to APD conversion).

Value Returned

t_newName	Appropriate class name based on product type.

Example 1

```
axlMapClassName("ETCH") -> "CONDUCTOR"
```

Gets APD class name when in APD.

Example 2

```
axlMapClassName("ETCH") -> "ETCH"
```

Gets Allegro PCB Editor class name when in Allegro PCB Editor.

axlMemSize

```
axlMemSize(
)
⇒ x_size
```

Description

Returns an estimate of memory use. Returns not what the program is *using*, rather what it is *requesting* from the operating system.

Argument

nothing

Value Returned

x_size	Estimate of memory use in bytes.

axlPPrint

```
axlPPrint(
     t_name
)
⇒ t_pname
```

Description

Converts a string with 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

`t`	PDF file displayed.
`nil`	Failed to display PDF file due one of the following: - No Acroread PDF viewer found on UNIX. - No PDF viewer registered on Windows.

Example

```
axlPdf("allegro.pdf")
```

axlRegexpIs

```
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_exp	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
     [?logfilet_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.

Argument

t_prog	Name of the program to run.
t_cmdFmt	Command string.
t_logFile	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.
startMsg	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...`"
reloadDB	If you set this to `t`, the database reloads after a successful run of the program. If the batch program does not save the database, you need not reload the database.

noUnload	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`.
silent	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`.
noProgress	If `t`, the progress meter does not display. Default is `nil`.
noLogview	When set, it prevents display if log file on program exit.
noWarnOnExit	When set, suppresses some exit warning messages.
warnProgram	Program supports warning status (returns 0, if success; 1, if warnings; and 2, if errors).
noExitMsgs	When set, suppresses messages about program success or failure.

Note: For `t_cmdFmt`, the formatting should include everything except the database file. Place a `%s` where the database should appear.

- To get a `%s` and still do a `sprintf` to fill in your optional command arguments, use a "`%%s`" as shown:

```
cmdFmt = "netrev -$ -q -r %s"
sprintf(cmdFmt, "%s -$ %s %s %%s", prog, argq argr)
```

Value Returned

`t`	Batch job ran.
x_error	Error number that is program dependent on failure.

Example 1

```
sprintf(format "genfeedformat -$ %s %%s", "-b")
axlRunBatchDBProgram("genfeedformat"
                     format
                     ?logfile "genfeed")
```

Spawns a `genfeedformat` which requires the database to be saved.

Example 2

```
axlRunBatchDBProgram("netin"
```

```
"netin -$ -g -y 1 netlist %s"
?startMsg "Logic Import"
?logfile "netin"
?reloadDB t)
```

Spawns `netin`, a program that requires read/write to the database.

Note: '`-$`' is a standard argument to Allegro PCB Editor batch programs that prevents prompting for missing input.

axlShowObject

```
axlShowObject(
     lud_dbid
)
⇒ t/nil
```

Description

Displays the object data for each *dbid* in *lud_dbid* in a *Show Element* window. Does the same function as the interactive command `list element.` *lud_dbid* can be either a single *dbid* or a list of *dbids*.

Arguments

lud_dbid	*dbid*, or list of *dbids*.

Value Returned

`t`	Displayed the **Show Element** window for any objects.
`nil`	Failed to display the Show Element window for any objects.

Example

```
axlDBCreatePropDictEntry(
     "myprop", "real", list( "pins" "nets" "symbols"),
     list( -50. 100), "level")
axlClearSelSet()
axlSetFindFilter(
     ?enabled '("NOALL" "ALLTYPES" "NAMEFORM")
     ?onButtons "ALLTYPES")
axlSingleSelectName( "NET" "ENA2")
axlDBAddProp(axlGetSelSet(), list("MYPROP" 23.5))
axlShowObject(axlGetSelSet())
⇒ t
```

1. Defines the string-valued property `MYPROP`.
2. Adds it to the net `ENA2`.
3. Displays the result to the user with `axlShowObject`.

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

x_time	Time in seconds.

Value Returned

`t`	All of the time.

axlSort

```
axlSort(
     t_infile
     t_outfile
     [t_sortfields]
     [t_sort_options]

)

⇒ t/nil
```

Description

Sorts contents of a given input file and places results in the output file. No warning is given if the output file overwrites an existing file - any checking must be done by the caller of this function.

Default sort is a left to right, ASCII ascending sort of the input file, with duplicate lines left in. You can control the sort behavior using the optional parameters.

Argument

t_infile	Name of the input file to sort.
t_outfile	Name of the file containing results of the sort.
t_sortfields	String specifying which fields to use as sort keys, the sort order of those fields, and how to interpret the data type of the field for sorting. String contains a series of triplets: `field_number sortOrder fieldType` String can contain multiple triplets separated by commas. Triplets can contain valid elements as shown:

Triplet Element	Description
field_number	Number representing the position of the field on the line, from left to right. Numbering starts at 1.
sortOrder	A - Ascending sort order
	D - Descending sort order
fieldType	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.

t_sort_options String containing directives controlling the global sort parameters. Sort options can appear in any order, and must be separated by commas. These options are available:

Option	Description
Field Delimiter	Supported delimiters include any character in punctuation character class except comma.
U	Remove duplicate lines. Default is keep duplicate lines.
D	Descending order. Default is ascending order.

An example of the *t_sort_options* string:

```
"!,U"
```

This means to use the '!' character as the field delimiter and to remove any duplicate lines.

Value Returned

`t` Sort successful.

`nil` Sort unsuccessful due to incorrect arguments.

Examples

Input file

```
2!3!4!5!6
2!3!4!5!6
5!6!7!8!9
5!1!7!8!9
2!2!3!4!5
1!2!3!4!5
3!4!5!6!7
4!5!6!7!8
```

Example 1

```
(axlSort "input.txt" "output.txt" nil "!,U")
```

Results 1

```
1!2!3!4!5
2!2!3!4!5
2!3!4!5!6
3!4!5!6!7
4!5!6!7!8
5!1!7!8!9
5!6!7!8!9
```

Example 2

```
(axlSort "input.txt" "output.txt" "2 A I, 1 D I" "!")
```

Results 2

```
5!1!7!8!9
2!2!3!4!5
1!2!3!4!5
2!3!4!5!6
2!3!4!5!6
3!4!5!6!7
4!5!6!7!8
5!6!7!8!9
```

axlStrcmpAlpNum

```
axlStrcmpAlpNum(
     t_str1
     t_str2
)
⇒ t/nil
```

Description

Provides an alpha-numeric sort similar to `alphalessp` with one important distinction. If both strings end in the number, the number portion is separated and the two stripped strings are first compared. If they are equal, then the number sections are compared as numbers, rather than strings.

`alphalessp` sort	`U1 U10 U2`
`axlStrcmpAlpNum` sort	`U1 U2 U10`

Arguments

`t_str1`	A string
`t_str2`	A string

Value Returned

`0`	The two strings are equal.
`+num`	If *`t_str1`* is greater than *`t_str2`* (1 goes after 2)
`nil`	If *`t_str1`* is less than *`t_str2`* (1 goes before 2)

Example

```
l = '("U5" "U10" "U1" "U5" "U2")
sort(l 'axlStrcmpAlpNum)
===> ("U1" "U2" "U5" "U5" "U10")
```

axlVersion

```
axlVersion(
     s_option
)
⇒ g_value/nil
```

Description

Returns Allegro PCB Editor or OS dependent data.

Argument

g_option	SKILL symbol for the environment variable name.

Value Returned

Return depends upon option given.

Option	Value	Returns
none	*ls_values*	List of available options.
version	*f_value*	Allegro PCB Editor program version, for example, 15.7.
tVersion	*t_value*	Allegro PCB Editor program version as a string, for example, 15.7.
fullVersion	*t_value*	Allegro PCB Editor program version and patch as a string (for example, 15.7 s01).
buildDate	*t_value*	Date program was built at Cadence (month/ day/year),for example, 7/19/2006.

Option	Value	Returns
`release`	*`t_value`*	Allegro PCB Editor release value. Consists of a prefix and a number (*<p><nn>*), where prefixes are as shown: A - Alpha B - Beta P - Production S - Patch For example, S23 indicates the 23rd patch release.
`internalVersion`	*`t_value`*	Internal program version, for example, v15-1-25A.
`programName`	*`t_value`*	Executable being run, for example, `allegro`.
`displayName`	*`t_value`*	Display name of program (may contain spaces.) Certain programs may change this during run-time depending on the license level.
`formalName`	*`t_value`*	Full formal name of program (may contain spaces). Certain programs may change during run-time depending on licensing level. The name can be the same or longer than *`displayName`*, and may change from release to release.
`isWindows`	*`g_value`*	`t` if Windows operating system. `nil` if UNIX operating system.
`isSQ`	*`g_value`*	`t` if Allegro PCB SI, `nil` otherwise.
`isPI`	*`g_value`*	`t` if Allegro PCB PI Option XL, `nil` otherwise.
`isAPDSi`	*`g_value`*	`t` if Allegro Package Designer SI L (or 610), `nil` otherwise
`isSQPkg`	*`g_value`*	`t` if SpecctraQuest for IC product, `nil` otherwise

Option	Value	Returns
`isSQPkg3D`	*g_value*	`t` if SpecctraQuest for IC product 3DC, `nil` otherwise
`isAllegroExpert`	*g_value*	`t` if Allegro PCB Design XL, `nil` otherwise.
`isAllegroDesigner`	*g_value*	`t` if Allegro PCB Performance L or Allegro PCB Design L product, `nil` otherwise.
`isAllegroPCB`	*g_value*	`t` if Allegro PCB product, `nil` otherwise.
`isAPD`	*g_value*	`t` if Allegro Package Designer L product, `nil` otherwise.
`isAPDL`	*g_value*	`t` if Allegro Package Designer L, `nil` otherwise
`isAPDXL`	*g_value*	`t` if Allegro Package Designer XL, `nil` otherwise
`isAPDLegacy`	*g_value*	`t` if Allegro Package Designer VT2300, `nil` otherwise
`isChipIOPlanner`	*g_value*	`t` if any `ChipIOPlanner` product, `nil` otherwise
`isSIP`	*g_value*	`t` if any SIP product, `nil` otherwise
`isSIPlayout`	*g_value*	`t` if SIP Layout product, `nil` otherwise.
`isSIPDigital`	*g_value*	`t` if any SIP Digital product, `nil` otherwise
`isSIPEngineer`	*g_value*	`t` if SIP Engineer product, `nil` otherwise.
`isSIPDigArch_L`	*g_value*	`t` if running the SIP Digital Architect L product, `nil` otherwise
`isSIPArchitect`	*g_value*	`t` if SIP Architect product, `nil` otherwise
`isSIPDigArch_GXL`	*g_value*	`t` if running the SIP Digital Architect GXL product , `nil` otherwise
`isSIPLibrarian`	*g_value*	`t` if SIP Librarian product, `nil` otherwise
`isSIPDigSI_XL`	*g_value*	`t` if running the SIP Digital SI XL product, `nil` otherwise
`isSIPDigLay_GXL`	*g_value*	`t` if running the SIP Digital Layout GXL product, `nil` otherwise

Option	Value	Returns
`isSIPRF`	*`g_value`*	`t` if running any SIP RF product, `nil` otherwise
`isSIPRFArch_L`	*`g_value`*	`t` if running the SIP RF Architect L product, `nil` otherwise
`isSIPRFLay_GXL`	*`g_value`*	`t` if running the SIP RF Layout GXL product, `nil` otherwise
`isSigxp`	*`g_value`*	`t` if any version of SigXP, `nil` otherwise.
`isSxExpert`	*`g_value`*	`t` if SigXP Expert product, `nil` otherwise.
`isSxExplorer`	*`g_value`*	`t` if SigXP Explorer product, `nil` otherwise.
`ICP`	*`g_value`*	`t` if an ICP based product (APD, SIP), `nil` otherwise.

Example

```
axlVersion()
```

axlVersionIdGet

```
axlVersionIdGet(
)
⇒ x_time
```

Description

Returns an id stamp based upon computer time.

Argument

none

Value Returned

x_time	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

x_time/*t_time*	VERSION_ID obtained from the database property or returned from `axlVersionIdGet()`.

Value Returned

t_printTime	Printable string in standard Allegro PCB Editor date/time format.
`nil`	Failed to print VERSION_ID due to incorrect argument.

Example

```
axlVersionIdPrint(axlDBGetDesign()->prop->VERSION_ID
-> "Mon Dec 16 12:45:16 2004"
```

24

Math Utility Functions

Overview

This chapter describes the AXL-SKILL Math Utility functions.

axlDistance

```
axlDistance(
     l_point1
     l_point2
     )

⇒ f_distance
```

Description

Returns the distance between two points. You may use floating point.

Arguments

l_point	A point.
ll_line	Two points at the ends of a line.

Value Returned

f_distance	Distance between two points in floating point form.

Example

```
axlDistance(10:20 5:20) = 5.0
```

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 then the value Allegro reports. This is difference is due to how floating point numbers are represented in the computer. The following article is an excellant 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

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 then 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

dbrep	a floating point number
point	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)
```

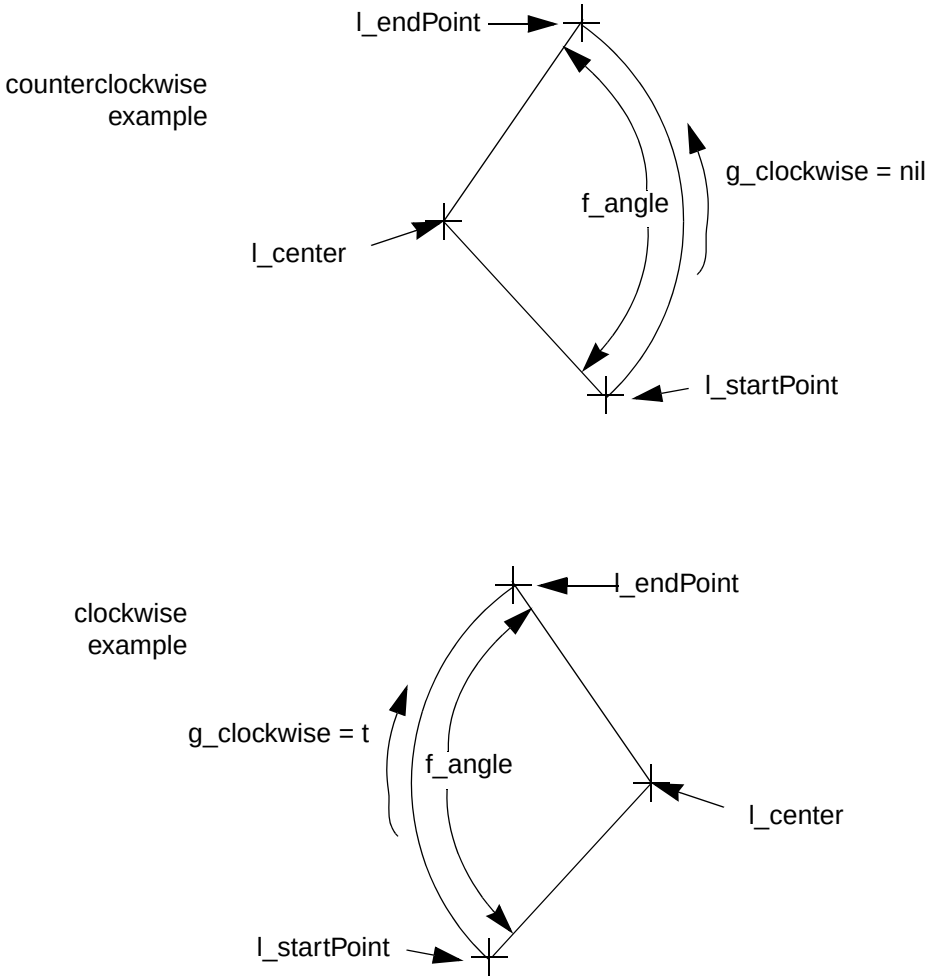
axlGeoArcCenterAngle

```
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 24-1 on page 1081.

Figure 24-1 Center of Arc Calculation



Arguments

l_startPoint	Start point of the arc
l_endPoint	End point of the arc
f_angle	Included angle of the arc
g_clockwise	Rotational sense of the arc: `t` is clockwise. `nil` is counterclockwise (the default).

Value Returned

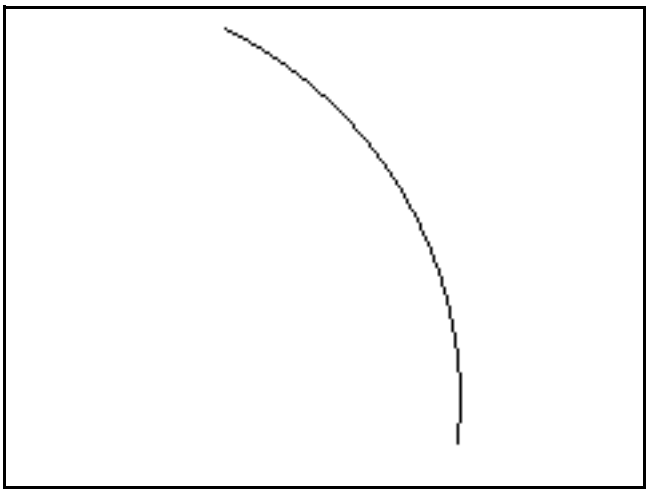
l_center	Center of the arc as a list: (*X Y*).
nil	Cannot calculate the center from the given arguments.

Example

```
print axlGeoArcCenterAngle( 7500:5600 8000:4700 68.5 t)
mypath = axlPathStart(list( 7500:5600))
     axlPathArcAngle(mypath, 0., 8000:4700, t, 68.5)
     axlDBCreatePath( mypath, "etch/bottom")
⇒(7089.086 4782.826)
```

Prints the center for the clockwise arc going through the points (7500:5600) and (8000:4700) using `axlGetArcCenterAngle`, then adds the arc through those points using `axlPathArcAngle`, and compares their centers.

The arc is shown in the following figure.



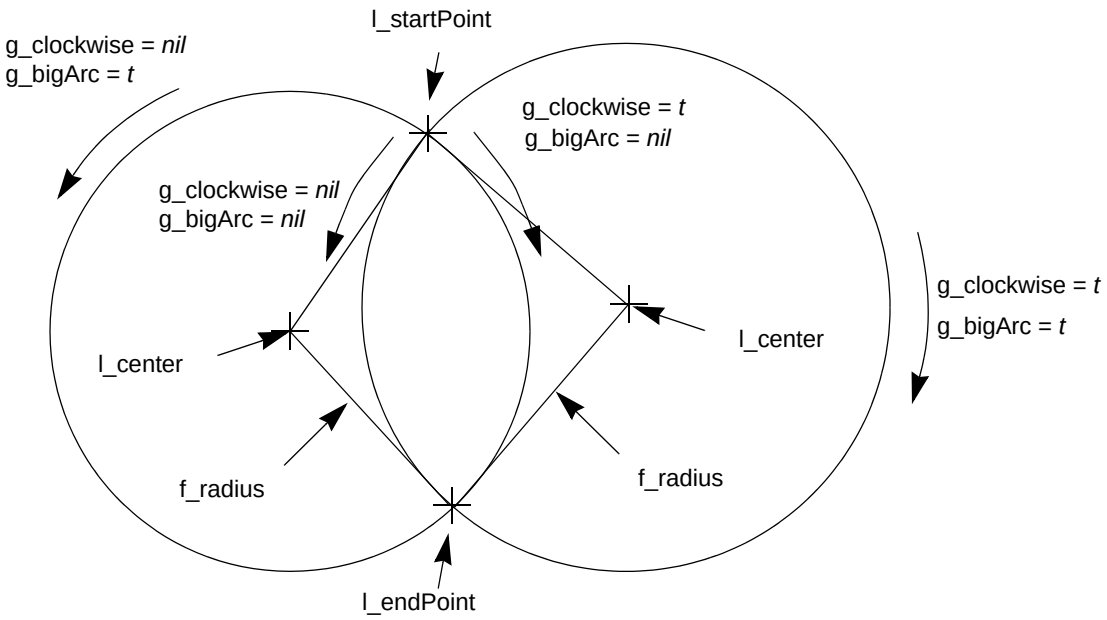
Do *Show Element* on the arc to show its center coordinate. The arc lists: `"center-xy (7089,4783)"` which agrees with the *(7089.086 4782.826)* printed by `axlGeoArcCenterRadius`.

axlGeoArcCenterRadius

```
axlGeoArcCenterRadius(
     l_startPoint
     l_endPoint
     f_radius
     [g_clockwise]
     [g_bigArc]
)
⇒ l_center/nil
```

Description

Calculates center of an arc given its radius. Calculates *l_center* either for one arc or another depending on the arguments, as shown.



Arguments

l_startPoint	Start point of the arc
l_endPoint	End point of the arc
f_radius	Radius of the arc

g_clockwise	Rotational sense of the arc: `t` is clockwise. `nil` (default) is counterclockwise.
g_bigArc	Flag telling whether the arc extends as the larger or the smaller of the two arcs possible between the start and endpoints.

Value Returned

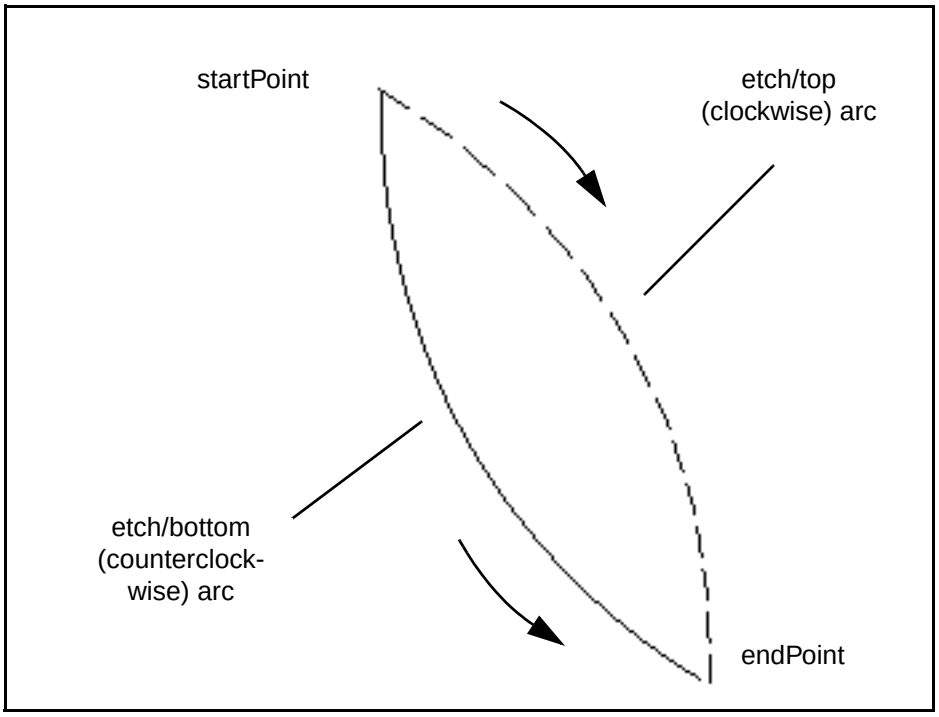
l_center	Center of the arc as a list: (*X Y*).
`nil`	Cannot calculate the center from the given arguments.

Example

```
print axlGeoArcCenterRadius( 7500:5600 8000:4700 1000)
⇒ (8499.434 5566.352)

mypath = axlPathStart(list( 7500:5600))
axlPathArcRadius(mypath, 0., 8000:4700, t, nil, 1000)
axlDBCreatePath( mypath, "etch/bottom")

print axlGeoArcCenterRadius( 7500:5600 8000:4700 1000 t)
⇒ (7000.566 4733.648)

mypath = axlPathStart(list( 7500:5600))
axlPathArcRadius(mypath, 0., 8000:4700, nil, nil, 1000)
axlDBCreatePath( mypath, "etch/top")
```

Prints the two possible centers for the arcs going through the points (7500:5600) and (8000:4700), then adds arcs through those points using `axlPathArcRadius`, and compares the centers.

The arcs are shown in the following figure.



Do *Show Element* on each arc to show its center coordinate.The solid arc on the left lists: `"center-xy (8499,5566)"` which agrees with the *(8499.434 5566.352)* printed by `axlGeoArcCenterRadius`. The dotted arc on the right lists: `"center-xy (7001, 4734)"`, which agrees within rounding with `(7000.566 4733.648)` printed for the other arc.

Arguments 3

```
axlMKSConvert(
     nil
     [t_outUnits]

)

⇒ t/nil
```

Pre-registers *t_outUnits*, the input units string, so that subsequent calls to axlMKSConvert need not specify units.

t_outUnits	String giving the units to convert to for subsequent calls to axlMKSConvert. 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

```
axlMKSConvert(
     n_input

)

⇒ f_output/nil
```

Use this combination of arguments only after a call to axlMKSConvert as in Arguments 3. Converts the number *n_input* specifying a value using the *t_outUnits* supplied by a previous call to axlMKSConvert, and returns as *f_output*.

n_input	Number giving the input value to convert

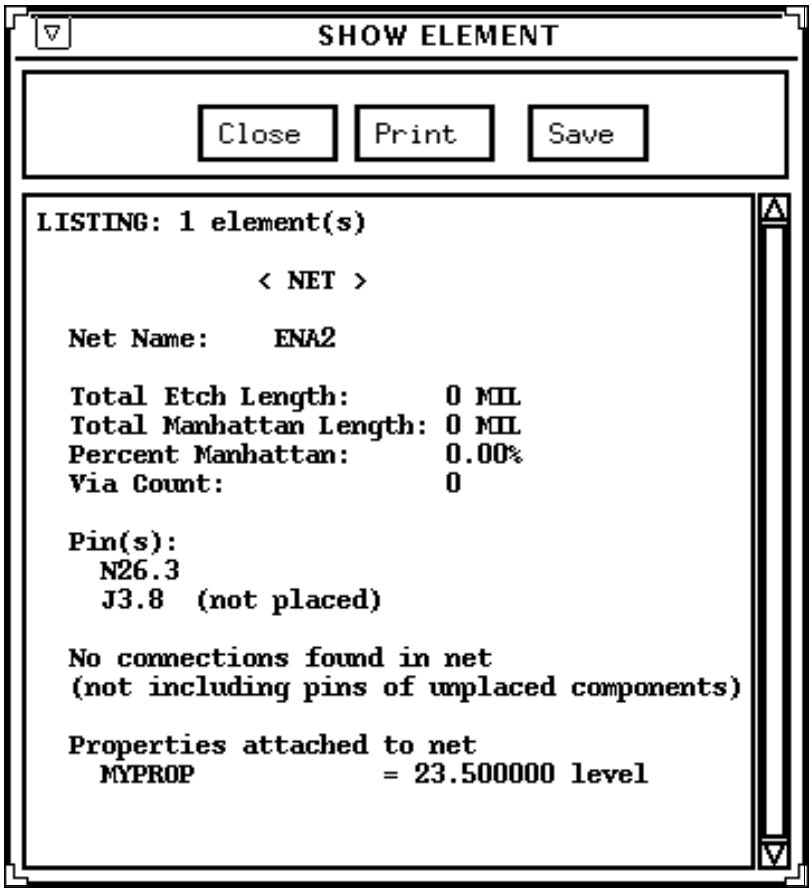
Value Returned 4

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:



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*."

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

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

f_angle	Angle 0 to 360 degrees (support for 1/1000 of a degree); rotation is counter-clockwise for positive numbers.
l_xy	xy point to rotate in user units.
l_origin	Origin to rotate about; if `nil` uses (0, 0) mirror: optional mirror flag (`t` perform mirror).

Value Returned

l_xyResult	Rotated result.
`nil`	Error in arguments or rotation results in return being outside of the database extents.

Examples

Rotate:

```
axlGeoRotatePt(45.0 10:200 5:2) ->   (-131.4716 145.5427)
```

Rotate and mirror:

```
axlGeoRotatePt(45.0 10:200 5:2 t) -> (-138.5427 138.4716)
```

Rotate about 0,0:

```
axlGeoRotatePt(90.0, 100:0 nil) ->  (0.0 100.0)
```

axlIsPointInsideBox

```
axlIsPointInsideBox(
     l_point
     l_box
     )
```

⇒ t/nil

Description

Returns `t` if a point is inside or on the edge of a box. Also see axlGeoPointInShape on page 1118 for *dbid*-based tests. You may use floating point.

Arguments

l_point	A point.
l_box	A bounding box.

Value Returned

`t`	Point is inside or on edge of box
`nil`	Point is outside of box.

Example

```
axlIsPointInsideBox(10:20 list(5:20 15:30)) = t
axlIsPointInsideBox(0:20 list(5:20 15:30)) = nil
axlIsPointInsideBox(15:20 list(5:20 15:30)) = t
```

axlIsPointOnLine

```
axlIsPointOnLine(
     l_point
     ll_line
     )
```

⇒ `t/nil`

Description

Returns `t` if point is on a given line or `nil` if not on the line. You may use floating point.

Arguments

`l_point`	A point.
`ll_line`	Two end points.

Value Returned

`t`	Point is on the specified line.
`nil`	Point is not on the specified line.

Example

```
axlIsPointOnLine(10:20 list(5:20 15:30)) = nil
axlIsPointOnLine(15:20 list(5:20 15:30)) = nil
```

axlLineSlope

```
axlLineSlope(
     ll_line
     )
```

⇒ *t_slope*/nil

Description

Returns the slope of a line. You may use floating point.

Arguments

ll_line	Two end points.

Value Returned

t_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
```

axlLineXLine

```
axlLineXLine(
     l_seg1
     l_seg2
     )

⇒ t
```

Description

This function is no longer required, but is kept for backward compatibility.

Arguments

None.

Value Returned

`t`	Returns `t` always.

axlMPythag

```
axlMPythag(
          l_pt1
          l_pt2
) -> f_distance/nil
```

Description

Calculates distance between two points using pythagoras. This is faster then building this code in Skill.

The `l_ptN` is an x:y coordinate.

Arguments

l_pt1, l_pt2	Two xy points.

Value Returned

`f_distance`	Distance between points.
`nil`	Arguments were not xy points.

See Also

`axlMXYAdd`

Example

```
axlMPythag(1263.0:1062.0 1338.0:1137.0) -> 106.066
```

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 trigometric 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)
```

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 + lpt2` 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)
```

axlMXYMultAdd

```
axlMXYMult(
                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 will add the orign.

```
(l_uniVec * f_factor) + l_origin
```

It is normally used in conjunction with `axlMUniVector` to project a point along a vector.

Arguments

l_uniVec:	xy point
f_factor:	multiplication factor
l_origin:	additive point

Value Returned

l_pt:	Returns resultant coordinate
nil:	Arguments are not coordinates

Examples

```
axlMXYMult(1263.0:1063.0 2.0) -> (2526.0 2126.0)
```

See Also

axlMUniVector

axlMXYSub

```
axlMXYSub(
          l_pt1
          l_pt2
) -> l_pt/nil
```

Description

This does a `l_pt1 - lpt2` and returns the result.

The `l_ptN` is an x:y coordinate.

Arguments

`l_pt1, l_pt2`	Two xy points.

Value Returned

`l_pt`	Returns coordinate that is result of subtraction.
`nil`	Arguments are not coordinates.

See Also

`axlMXYAdd`

Example

```
axlMXYSub('(1263.0 1063.0) '(1338.0 1137.0)) -> (-75.0 -74.0)
```

axl_ol_ol2

```
axl_ol_ol2(
     l_seg1
     l_seg2
     )

⇒ l_result
```

Description

Finds the intersection point of two lines. If the lines intersect, returns the intersection point with a distance of 0. If the lines do not intersect, the distance is not zero and the function returns `t`.

Arguments

l_seg1	1st line segment (list *x1*:*y1 x2*:*y2*)
l_seg2	2nd line segment (list *x1*:*y1 x2*:*y2*)

Value Returned

`nil`	Error due to incorrect argument.
(*car l_result*)	Intersect or nearest point on seg1
(*cadr l_result*)	Intersect or nearest point on seg2
(*caddr l_result*)	Distance between the two intersect points.

Examples

Data for examples

```
a=list(1:5 5:5)
b=list(2:5 4:2)
c=list(0:0 5:0)
d=list(4:5 7:5)
```

Example 1

```
axl_ol_ol2(a b)
```

⇒((2.0 5.0) (2.0 5.0) 0.0)

Intersects line, returns intersection point, note distance of 0.

Example 2

```
axl_ol_ol2(a c)
```

⇒((3.0 5.0) (3.0 0.0) 0.0)

Lines don’t intersect, returns closest point on each line and distance.

Example 3

```
axl_ol_ol2(a d)
```

⇒((4.5 5.0) (4.5 5.0) 0.0)

Lines overlap, distance of 0 and selects mid-point of the overlap.

bBoxAdd

```
bBoxAdd(
           l_bBox1
           l_bBox2
          ) -> l_bBox_result
```

Description

This addes 2 bBox together and returns the result

Arguments

2 Bboxs

Value Returned

Resulting bBox

Example

Expand bounding box by 100

```
orig = '((200 100) (400 500))
res = bBoxAdd(orig '((-100 -100) (100 100)))
-> ((100 0) (500 600))
```

25

Database Miscellaneous Functions

Overview

This chapter describes the AXL-SKILL functions that don’t fit into other sections.

axlAirGap

```
axlAirGap(
         o_item1DBID
         o_item2DBID
         [t_layer]/nil
         [s_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 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 comparision without the layer, we use the current active layer. 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".

Output data appears in one of the following formats depending on the `s_mode` option:

- 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:

`t`	`Success (t (l_airGapData))`
`'NOMATCH`	No subclass matches between pin or via and object. Returns object's layer. (`NOMATCH (t_layer)`)

'RANGE

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 `nil`: (`ETCH (t_topMatch t_bottomMatch)`)

'INVALID

One or both elements are invalid. Data return format: (`INVALID nil`). 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 in any layer, use `s_mode = 'anyLayer`.

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.

Arguments

o_item1DBID	`dbid` of the first item.
o_item2DBID	`dbid` of the second item.
t_layer	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 comparson. It is most useful in measuring mask layer gaps.
s_mode	Return additional info to clarify error. This may be: ■ `nil`: Default mode (objects must be on same layer) ■ `t`: `("full mode") return l_airGapData` or if not share see above (objects must be on same layer) ■ `anyLayer`: support any layer measure return just gap ■ `enhanced`: return disembodied property list of additional air gap criteria (see above)

Value Returned

l_airGapData	List containing the following items: `(l_airGapPt1 l_airGapPt2 f_airGapDistance)`

where:

l_airGapPt1	(X,Y) point on the first item where the air gap is measured.
l_airGapPt2	(X,Y) point on the second item where the air gap is measured.
f_airGapDistance	Distance between the two points.

nil	Input data error; element 1 and 2 are the same or no air gap can be computed between the two items. If `t_layer` is used but does not specify an etch layer.
s_error	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(via1 pin2 ""PIN/SOLDERMASK_TOP" )
-> (((1337.5 1100.0) (1362.5 1100.0) 40.0))
```

Full output failure:

```
axlAirGap(el3 el2 nil t)
-> (RANGE ("TOP" "GND"))
```

axlBackDrill

```
axlBackDrill(
          o_dbid
          s_layer
           ) -> l_result/nil
```

Description

pin/via backdrill analysis

Does a backdrill analysis on a given pin or a via (`o_dbid`) where the backdrill should start on top or bottom (`s_layer`).

Note: This is a tier limited feature and is therefore, not available with all versions of the tool.

This analysis is based upon the current backdrill parameter settings.

(see backdrill setup command).

Result of analysis is a disembodied property list containing:

Symbol	Type	Data
status	symbol	result of analysis (see below)
toLayer	number	xsection layer number for end drill
remainStub	float	lengthStub - depthDrill
lengthStub	float	depth (user units) of desired drill
depthDrill	float	depth (user units) of actual drill
maxStub	float	max stub from net's BACKDRILL_MAX_PTH_STUB property or 0 if no prop
minPth	float	pin/via or symbol's BACKDRILL_MIN_PIN_PTH property value or 0 if no prop

Status results

Skip Drill Items due to:

net	net does not have the BACKDRILL_MAX_PTH_STUB property or item not on a net
stubOk	stub is ok to skip (drill_depth < maxStub)
noStub	no stub on item
skipThisSide	drilling not permitted on this side of design
pad	skip because this object type (pin or via) should not be drilled
holeType	padstack has no hole, non-plated or a slot

Exclude Drill Items

via	via has BACKDRILL_EXCLUDE property
pin	pin has BACKDRILL_EXCLUDE property
symbol	symbol has BACKDRILL_EXCLUDE property
unconnected	pin/via has no connections
testVia	via is a test via
testPin	pin is a test pin
excludePinSide	exclude pin side due to BACKDRILL_PRESSFIT_CONNECTOR on symbol

Errors

Argument	**Valid Values**
errorStubLen	Drill depth exceeds BACKDRILL_MAX_PTH_STUB value
errorPinPth	(boardThickness - drillDepth) of pin is less the property value BACKDRILL_MIN_PIN_PTH for pressfit connectors.
unknown	unknown problem (atypical)

Arguments

Argument	Valid Values
`o_dbid`	■ pin ■ via
`s_layer`	■ top ■ bottom

Value Returned

`nil`	Indicates an error, generated due to one of the following reasons ■ This feature is not available in this editor ■ argument specified is not a pin or a via
`l_result`	a disembody property list see above for desription

Example

```
result = axlBackDrill(pin, 'bottom)
```

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 Manhatten length. If a net is partially routed, includes sum of all ratsnest Manhatten 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`

Example

```
Skill> p = ashOne()
Skill> axlDBGetLength(p)
-> 2676.777
```

axlDBGetManhattan

```
axlDBGetManhattan(
     o_dbid_net
)
⇒ l_result/nil
```

Description

Given a net, calculates an etch, path, and Manhattan length. The result is the same as that used by list element.

- Etch - The current length of etch. The length is 0 when there is no etch.
- Path - The etch plus remaining length. When the net is fully connected, there is no remaining, and path is equal to etch.
- Manhattan - The estimated routing length.

Note: Path is equal to Manhattan when the net has no etch.

Arguments

o_dbid	Net *dbid*.

Value Returned

l_result	(*etchLength path manhattan*)
nil	Not a net *dbid*. Net is out of date. No ratsnest.

See Also

axlDBGetLength

Example

```
p = ashOne()
axlDBGetManhattan(p)
(2676.777 3300.0)
```

axlExtentDB

```
axlExtentDB(
)
⇒ l_bBox/nil
```

Description

Determines a design type and returns the *bBox* extent. See axlExtentLayout and axlExtentSymbol for what Allegro PCB Editor considers an extent.

Arguments

None

Valued Returned

l_bBox	Returns *bBox* extent.
nil	Unknown drawing type.

axlExtentLayout

```
axlExtentLayout(
)
⇒ l_bBox/nil
```

Description

Obsolete. Use `axlExtentDB`. Kept for backward compatibility.

Computes the layout extents and returns the smallest bounding box to be used for window-fit. Only `lines`, `linesegs`, and `shapes` are searched on selected layers in the following order:

1. `BOARD GEOMETRY/OUTLINE`
2. `PACKAGE KEEPIN/ALL`
3. `ROUTE KEEPIN/ALL`

The first layer with any elements is used to determine the layout extents. If no elements are found on these layers, the design extents are returned.

Arguments

None

Value Returned

l_bBox	Returns *bBox* of the layout. (See `axlExtentDB`.)
`nil`	Error such as the failure of `axlVisibleGet`.

See Also

`axlExtentDB`

axlExtentSymbol

```
axlExtentSymbol(
)
⇒ l_bBox
```

Description

Obsolete. Use `axlExtentDB`. Kept for backward compatibility.

Computes the bounding box enclosing all objects visible for a drawing (a `.dra` file).

Arguments

None

Value Returned

l_bBox	Smallest bounding box enclosing all visible objects. If no objects are visible, set to the design extents. (See `axlExtentDB`.)

See Also

`axlExtentDB`

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

l_point	Point to check.
o_dbid/o_polygon	dbid of the shape / o_polygon
[g_include_voids]	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 t.
[t/nil]	t means include voids, nil means use the shape outline only. Default is t.

Value Returned

t	Point is inside the shape.
nil	Point is outside the shape, or incorrect arguments were given.

See Also: axlGeoPointShapeInfo

axlGeoPointShapeInfo

axlGeoPointShapeInfo - relation of a point to a shape

```
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

o_dbid	Segment cline

Value Returned

f_min f_max	Impedance in current MKS units.
nil	Segment is not a cline segment.

See Also

axlSegDelayAndZ0

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`

`axlImpdedanceGetLayerEdgeDPImp`

`axlImpdedanceGetLayerEdgeDPWidth`

`axlImpdedanceGetLayerEdgeDPSpacing`

`axlImpdedanceGetLayerBroadsideDPImp`

`axlImpdedanceGetLayerBroadsideDPWidth`

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 or antipads as Route Keepouts (ARK) via the g_type optins.

`'type`

Changes a bbvia padstack to a micro via and vice versa. Uvia types can be managed seperately in the contrainst system.This has no effect if the padstack is used with Pins. Values are '`bbvia` or '`uvia`.

`'keepout`

Enables (or disables) the antipads as Route Keepouts. This indicates the padstack has been built to allow use of the antipad to generate the equivalent of a rko for mechanical pins. It has no effect if these padstacks are used for logical connections. Values are `t` or `nil`.

Marks DRC out-of-date if successful.

Arguments

o_padstack	padstack dbid
t_padstack	padstack name

g_type	mode (either 'type or 'keepout)
g_value	appropriate setting (see above)

Value Returned

t	change successful
nil	failed. Not a padstack, padstack not in database, type not recognized or padstack not a bbvia or uvia.

Examples

Change padstack named VIA to a micro via

```
axlPadstackSetType("VIA" 'type 'uvia)
```

Change to support antipads as Route Keepouts (ARK)

```
axlPadstackSetType("VIA" 'keepout t)
```

See Also

axlDBCreatePadStack

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 imdepdance

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_AREA`	Testpoint pad in `NO_PROBE` area
`PIN_IS_VIA`	Pin type requires a via or any point
`PIN_NOT_VIA`	Pin type requires a via
`PIN_NOT_OUTPUT`	Pin type requires an output pin for test point
`PIN_NOT_IO`	Pin type requires an IOpin for test point
`PIN_TOO_CLOSE`	Pin too close to another test point
`PAD_UNDER_PIN`	Test point under another pin
`PIN_NOT_NODE`	Test point requires a node for testbench
`FIXED_TEST_POINTS`	`Testpoints are fixed and cannot be removed`
`OTHER`	Unclassified error

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 'info1 "Select pin or via to add testpoint")
dbid = ashOne('(VIAS PINS))
ret = axlTestPoint(dbid 'top)
```

2) Clear a testpoint

```
axlUIWPrint(nil 'info1 "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 (frectangles), 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.

Arguments

o_dbid	Shape *dbid*
t_netName/o_netdbid	Name of a net or a netdbid (for dummy nets)

Value Returned

t	Object changed.
nil	No object changed.

axlSegDelayAndZ0

```
axlSegDelayAndZ0(
     o_clineSegDbid
)
⇒ (f_delay f_z0)/nil
```

Description

Returns the delay and impedance of a cline segment. Returns `nil` if a segment isn't a cline segment. Normally, delay is in nanoseconds and impedance is in ohms.

This function is noisy if you pass in non-cline segments.

Arguments

o_clineSegDbid	Segment cline

Value Returned

f_delay f_z0	Delay and impedance in current MKS units.
`nil`	Segment is not a cline segment.

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 a MCM or SIP 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.

26

Plugin Functions

Overview

This chapter describes the AXL-SKILL functions related to plugin APIs.

The axlDll family of APIs provides the ability to bind compiled Dll (shared library) packages into programs supporting the Skill axl APIs. Any publically 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 progamming model for plugins is:

- Locate and open a plugin via `axlDllOpen`. We suggest assigning a handle to a global Skill symbol. On subsequent calls into your Skill application, the symbol will be non-nil, so the call to `axlDllOpen` can be skipped. (See `axlDllOpen`.)
- Import the required symbols from the plugin via `axlDLLSym`. Like the handle returned by `axlDllOpen`, these handles should be assigned to global symbols.
- Use either `axlDllCall` or `axlDllCallList` to access the capability from the plugin.
- The I/O data types that are supported are documented in `axlDllCall`.

DLL Programming

DLL programming allows the use of external developers to utilize existing C/C++ code or devevlop 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:

- 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:

`return == 0`	Returns a list of from output data structure up to the count value. If count in output structure is 0, returns an empty list.
`return != 0`	Return value is returned to Skill as an integer.

AXLPluginArgs Input/Output Structure Members

`version = AXLPLUGIN_VERS_IN`	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.
`flag = 0`	Future use
`maxEntries = AXLPLUGIN_MAX`	Can be ignored on input. On ouput, you cannot exceed this value for return arguments.
`count = <number of argv entries>`	Number of entries in `argv`. Basically, the number of arguments provided to the `axlDLLCall` APIs. Will never be more the `maxEntries`.
`argv[]`	Array of `AXLPluginEntry` arguments

AXLPluginEntry (argv) Structure Members

(See Input/Output Data Primitives.)

`type`	Indicates type of data
`data`	Union of primitive types supported

For the called Dll function to return data to calling Skill code, both `set the count` and `argv` entries should be set in the output data structure. For each `argv` entry return, you should also set the data type to one of the primitives shown below. Under no situation should you attempt to return more then `maxEntries`.

If your exported dll functions will be used outside your progamming 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` reprensents 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 the 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 dependancies 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:

  - Window: `depends` or `procexp` (`www.sysinternals.com`)
  - Solaris, Linux: `ldd <program name>`
  - AIX: `dump -H <program name>`
- On Windows, you can include UI components in your plugin. This capability is not supported on Unix.
- Wide character types are not supported in the plugin to the Allegro interface.
- On Unix, if threading is used, then you should `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 impatible 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 supporting 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 (UNIX) and project file (Visual.NET) is contained at `<cdsroot>/share/pcb/examples/plugin`. In addition, a prebuilt 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.
- A `x_value` API which returns the number of input arguments.
- A distance implementation to calculate the distance between two points.

axlDllCall

```
axlDllCall(
          o_pluginFunc
          [g_arg1]
          ...
          [g_arg10]
) -> nil/x_value/lg_data
```

Description

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 `axlDllSym`. 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 it calls the import function from the plugin

Arguments

`o_pluginFunc`	plugin symbol handle
`[g_arg1..10]`	up to 10 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

axlDllOpen

Example

Example carried from `axlDllSym`

```
axlDllCall(_ashDistance 10:0 25.1:0) -> 15.1
```

axlDllCallList

```
axlDllCall(
          o_pluginFunc
          l_args/nil
) -> nil/x_value/lg_data
```

Description

This function is identical to `axlDllCall` except it takes a list of arguments to pass to the plugin function. Unlike `axlDllCall`, which is limited to 10 arguments, this interface can take up to 512 arguments.

See `axlDllCall` 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

`axlDllCall`, `axlDllOpen`

Example

From `axlDllOpen` example

```
_ashEcho = axlDllSym(_ashTestDll "ashEcho")
```

```
axlDllCallList(_ashEcho list(-1 -2.0 nil "another string"
          -10.1:-2 0.2))
```

axlDllClose

```
axlDllClose(
          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 `axlDllOpen`.

Value Returned

`t` if successful to close, `nil` if handle is not a legal handle

See Also

`axlDllOpen`

Example

See example referenced by `axlDllOpen`.

axlDllDump

```
axlDllDump(
)
==> 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

`axlDllOpen`

Example

```
axlDllDump()
```

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 1140 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 (`axlDllSym`).

Plugin Attributes

Name	Type	Description
name	string	Name of plugin file (dll name)
functions	l_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 indendence, 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 1140, axlDllSym, axlDllCall, axlDllCallList axlDllClose, axlDllDump

Example

Open Cadence test dll

```
_ashTestDll = axlDllOpen("axlecho_plugin")
```

axlDllSym

```
axlDllSym(
          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 `axlDllDoc`).

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 `axlDllOpen`
`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

`axlDllOpen`

Example

Load the distance symbol from the axl plugin test dll

```
_ashDistance = axlDllSym(_ashTestDll "ashDistance")
```

27

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 provdes 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 betgween 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 termintes 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 bindins 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 `axldo`, and serially for `axldoStar`.

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 conditional 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

`g_initList`	0 or more index variable specifiers
`g_termiateList`	end test predicate
`g_body`	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 orginal 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 mprocedure function.

Arguments

`l_bindings`		list of l_varbind where:
	`l_varbind`	is (<name> <value>) where
	`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.

Example

```
(listnindex "dog" '("three" "dog" 'night)) -> 2
```

movedown

movedown - move an element one item farther from the head of a list

```
movedown(
          g_elem
          l_list
        ) --> l_newlist
```

Description

Find all occurences of element within list `l` and move 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 occurences of the element within list `l` and moves them one item closer to the list head. No action for other items and the car.

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

```
s_handler (
     l_curLineInfo
      l_lineArgs
      g_result
      )
      ==> g_result
```

Where:

`l_curLineInfo` List of info which describes the current line being processed.

Defined as:

`(t_fileName g_lineNo t_curString)`

`t_fileName` Name of file being read

`g_lineNo` Current line number

	`t_curString`	Unparsed file line
`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 reult 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 charanter 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)
```

28

Logic Access Functions

Overview

This chapter describes the AXL-SKILL functions related to schematic capture or the logical definition of the design.

axlDBCreateConceptComponent

```
axlDBCreateConceptComponent(
     s_refdes
     s_partPath
     s_logName
     s_primName
     [s_pptRowName]
)
⇒ r_dbid/nil
```

Description

Given the Concept information needed to describe an 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 `cptList`*XXX* family of routines, which allow browsing through a Concept library.

Arguments

s_refDes	Reference designator for the new component.
s_partPath	Full path to the `chips_ptr` file containing the description of the desired logical part. Determine using the `cptListComponentLibraries` function.
s_logName	Name of the desired logical part. You can determine this using the `cptListComponentPrimitives` function.
s_primName	Name of the desired primitive. You can determine this using the `cptListComponentPrimitives` function.
s_pptRowName	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 `cptListComponentDevices()` function.

Value Returned

r_dbid	*dbid* of the new Allegro PCB Editor component instance.
nil	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.

axlDBCreateComponent

```
axlDBCreateConceptComponent(
             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

`s_refDes`	The reference designator for the new component.
`s_deviceName`	Name of Allegro PCB Editor device file.
`s_package`	Package name to be used for the component. This overrides the value found in the device file (can be nil).
`s_value`	"Value" attribute value. This will override the value found in the device file (can be nil).
`s_tolerance`	"Tolerance" attribute value. This will override the value found in the device file (can be nil).

Value Returned

`r_dbid`	`dbid` of new Allegro PCB Editor component.
`nil`	If unable to create component.

Note: If you change an existing component definition by specifying a new value for package, value, or tolerance, you need a device file.

axlDBCreateManyModuleInstances

```
axlDBCreateManyModuleInstances(
             t_name
             t_moddefName
             x_tileStartNum
             l_origin
             l_offset
             x_num_tiles
             f_rotation
             x_logicMethod
             [l_netExcept]
               [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

t_name	Prefix of the names of the module instances.
t_moddefName	Name of the module definition/
x_tileStartNum	Tile numbering start number and increment by 1 for each tile.
l_origin	First location of first module.
l_offset	List containing the offset wanted between module origins.
x_numTiles	The number of module instances to be place.
f_rotation	Angle of rotation for the module instance.
x_logicMethod	Flag to indicate where the logic for the module comes from.

`0`	No logic.
`1`	Logic from schematic.
`2`	Logic from module definition.
`l_netExcept`	Optional list of net names to add to the net exception list.
`g_mirror`	Optional if modules should be mirrored

Value Returned

`lo_result`	If successful, returns the list of database objects that belong to the module instances created.
`nil`	Creation of module instance could not be completed.

See Also

axlDBCreateModuleDef, `axlDBCreateModuleInstance`

Example

Add five module instances based on the module definition file `mod.mdd`, starting at point (`10,10`) and offsetting by (`5,0`) every time:

```
modinsts = axlDBCreateManyModuleInstances(
"Num" "mod" 2 10:10 '(5 0) 5 0 2 '("GND" "+5"))
```

Creates five module instances named `Num2`, `Num3`, `Num4`, `Num5`, `Num6`.

axlDBCreateModuleDef

```
axlDBCreateModuleDef(
     t_name
     l_origin
     l_objects
)
⇒ t/nil
```

Description

Creates a module based on existing database objects.

Arguments

t_name	String providing the name of the module definition. File name is the module definition name appended with `.mdd`.
l_origin	Coordinate serving as the origin of the module definition.
l_objects	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")
comp1 = car(axlGetSelSet())
axlSingleSelectName("COMPONENT" "U2")
comp2 = car(axlGetSelSet())
axlDBCreateModuleDef("comps" '(0 0) '(comp1 comp2))
⇒ A module definition file named comps.mdd is created.
```

Creates a module definition file containing two components.

axlDBCreateModuleInstance

```
axlDBCreateModuleInstance(
     t_name
     t_moddef_name
     l_origin
     r_rotation
     i_logic_method
     l_net_except
)
⇒ o_result/nil
```

Description

Allows you to use or place a previously defined module.

Arguments

t_name	String providing name of the module instance.
t_moddef_name	String providing name of the module definition to base the instance on.
l_origin	Coordinate location to place the origin of the module definition.
r_rotation	Angle of rotation for the module instance.
i_logic_method	Flag indicating where the logic for the module comes from: 0 - no logic 1 - logic from schematic 2 - logic from module definition.
l_net_except	Optional list of net names to add to net exception list.

Value Returned

o_result	Database object that is the group used to represent the module instance.
nil	Module instance not created.

Example

```
modinst = axlDBCreateModuleInstance("inst" "mod" '(500 1500) 2 '("GND" "+5"))
```

Adds a module instance based on the module definition file mod.mdd.

axlDBCreateSymDefSkeleton

```
axlDBCreateSymDefSkeleton(
            l_symbolData
            l_extents
            [l_pinData]
        )
        ==> axlDBID/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 `axlDBCreate` 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

l_symbolData	A list of (`t_symbolName [t_symbolType]`).	
	t_symbolName	Name of the symbol.
	t_symbolType	Package (default), mechanical, or format.
l_extents	The lower left and upper right corners of the symbol def extents.	
l_pinData	List of `axlPinData` defstructs for the pins.	

Value Returned

`axlDBCreateSymDefSkeleton` `nil` if not created, or `axlDBID` of the symbol definition.

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.

axlDbidName

```
axlDbidName(
                  o_dbid
        ) -> t_name/nil
```

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, `axlDBGetDesign` 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()
axlDbidName(pin)
-> "U1.1"
```

Cline name:

```
cline = ashOne()
axlDbidName(cline)
-> "Net3, Etch/Top"
```

axlDiffPair

- Add DiffPair

```
axlDiffPair(
     t_diffpair
     o_net1/t_net1
     o_net2/t_net2

)

⇒ o_diffpair/nil
```

- Modify DiffPair

```
axlDiffPair(
     o_diffpair/t_diffpair
     o_net1/t_net1
     o_net2/t_net2

)

⇒ o_diffpair/nil
```

- Delete DiffPair

```
axlDiffPair(
     o_diffpair/t_diffpair

)

⇒ t/nil
```

Description

Creates, modifies, or deletes a differential pair. In all cases you can pass names or a *dbid*.

Note: If the differential pair was created due to Signoise models, you cannot modify or delete it. Consequently, you cannot modify or delete the differential pair if the following is true:

```
diffpair_dbid->prop->DIFFP_ELECTRICAL ==t.
```

Arguments

o_diffpair	Diffpair *dbid*
t_diffpair	Diffpair name.
o_net	Net *dbid*.
t_net	Net name.

Value Returned

Values returned depend on whether adding, modifying, or deleting.

o_diffpair	*dbid* of new or modified diffpair
t	Diffpair deleted.
nil	Error due to incorrect arguments.

Example 1

```
DPdbid = axlDiffPair("DP" "NET1+" "NET2-")
```

Creates a differential pair and names it DP1.

Example 2

```
DPdbid = axlDiffPair(DPdbid "NET1+" "NET1-")
```

Modifies a differential pair.

Example 3

```
axlDiffPair(DPdbid)
```

Deletes a differential pair.

Example 4

```
axlDiffPair(axlDBGetDesign()->diffpair)
```

Delete all differential pairs in design.

axlDiffPairAuto

```
axlDiffPairAuto(
     t_diffPairPrefix
     t_posNetPostfix
     t_negNetPostfix
     [g_returnDiffPairList]

)

⇒ x_cnt/(xcnt lo_diffpair)/nil
```

Description

Allows automatic generation of the diffpair. Generates the set of diffpairs based on the provided positive (*t_posNetPostfix*) and negative (*t_negNetPostfix*) postfixes used in your net naming.

You may provide a prefix (*t_diffPairPrefix*) used in generating the diffpair names of the form: <*t_diffPairPrefix*> + *netname* - *postfix*.

If nets are part of busses and end the bitfield syntax (<1>), the syntax portion is ignored when performing suffix matching. If a diffpair is created the bit number is added to the base net name used in forming a diffpair. For example, given two nets; DATA_P<1> and DATA_N<1> will result in a diffpair called DP_DATA1 with this call:

```
axlDiffPairAuto("DP_" "_P" "_N")
```

Arguments

t_diffPairPrefix	String to prefix diffpair names. Use "" if no prefix is desired.
t_posNetPostfix	Postfixes used to identify + diffpair members.
t_negNetPostfix	Postfixes used to identify - diffpair members.
g_returnDiffPairList	Controls return.

Value Returned

Returns depend on value of *g_returnDiffPairList* as shown:

g_returnDiffPairList	Value Returned	Description
`nil`	*`x_cnt`*	Number of diffpairs created.
`t`	(*`x_cnt`*, *`lo_diffpair`*)	List of diffpairs created

Examples

Example nets

Two nets called `NET1+` and `NET1-` are passed to this function.

Example 1

```
axlDiffPairAuto("DP_" "+" "-")
```

Shows diffpair creation and name generation. Results in one diffpair called `DP_NET1` with members `NET1+` and `NET1-`.

Example 2

```
axlDiffPairAuto("" "+" "-")
```

Gives the same result as the previous example, but names the diffpair `NET1`.

axlDiffPairDBID

```
axlDiffPairDBID(
     t_name
)
⇒ o_dbid/nil
```

Description

Returns the *dbid* of the named diffpair (*t_name*) if it exists in the database.

Arguments

t_name	Diffpair name.

Value Returned

o_dbid	*dbid* of diffpair if it exists.
nil	Diffpair does not exist.

axlDBGetExtents

```
axlDBGetExtents(o_dbid
                        g_visibleOnly
        )
        ==> bBox/nil
```

Description

Provides the extents of a physical database object. You choose between getting the extents of the entire object (even if it is currently not visible) or just the current visible objects.

With the visible option the extents may be smaller then the `bBox` attribute of the element `dbid bBox`.

If a list of `dbids` is provided, the union of the extents is returned.

Arguments

o_dbid	`dbid` of an element.
lo_dbid	list of `dbids`
visible_only	`t` return the extents of visible parts of the element. `nil` return the extents of the entire element.

Value Returned

bBox	The extents of the element. This might be zero extents, if the object has no extents visable or does not have extents (for example, nets).
`nil`	Error; bad arguments

axlMatchGroupAdd

```
axlMatchGroupAdd(
                  o_mgdbid/t_mgName
                  o_dbid/lo_dbid
         )==> t/nil
```

Description

Adds members to a matched group. Eligible members are:

- nets (if a net is part of an xnet the xnet is added to the group)
- xnets
- pinpairs

See discussion in `axlDBMatchGroupCreate`.

Command fails in product tiers that do not support electrical constraints or the symbol editor.

Tip

Using `dbids` is faster than using names.

Arguments

o_mgdbid	`dbid` of a match group.
t_mgName	Name of a match group.
o_dbid	Legal database `dbid` to add to match group.
lo_dbid	List of legal database `dbids` to add to match group.

Value Returned

`t`	Added elements.
`nil`	Failed one or more element additions.

See Also

axlMatchGroupCreate

Example

Add two nets to match group created in `axlMatchGroupCreate`:

```
mg = car(axlSelectByName("MATCH_GROUP" "MG1"))
nets = axlSelectByName("NET" '("B1_OUT" "B2_OUT"))
axlMatchGroupAdd(mg  nets)
```

axlMatchGroupCreate

```
axlMatchGroupCreate(
                  t_name
        )==> o_mgdbid
```

Description

Creates a new match group. If a match group already exists with the same name, `nil` is returned. Match groups need to be populated, or they are deleted when saving. Command fails in product tiers that do not support electrical constraints or the symbol editor.

If the match group was partially or completely created from an ECset, you can delete it, but it reappears when ECset flattening is required due to modifications in the design. SKILL functions do not indicate ECset-derived match groups.

RELATIVE_PROPAGATION_DELAY

The RELATIVE_PROPAGATION_DELAY proeprty can be added to the match group, xnet and pinpairs. If you add it to the match group then any match group member that does not have the property inherits it from the match group. Match groups contain the following elements: xnets, nets, and pinpairs. If a net is part of an xnet, the xnet is added to the match group. Xnets and pinpairs can belong to multiple match groups, so an RPD property exists on the `dbid` for nets, xnets and pinpairs. This property is a list of lists where each sub-list contains a match group `dbid` and the RELATIVE_PROPAGATION_DELAY value:

```
rpd = ( (o_mgDbid t_rpdValue) ....)
```

If a pinpair belongs to multiple match groups, you see two lists. In cases where the `dbid` belongs to a match group but has no `rpd` value, the *`t_rpdValue`* reports a `nil`.

You can add and delete properties to a match group or pinpair `dbid` using `axlMatchGroupProp`. When creating the property value, you must include the match group name but not an explicit pinpair. For example, the following adds an RPD property to a match group named `MG2`:

```
axlDBAddProp(mg '("RELATIVE_PROPAGATION_DELAY" "MG2:G:::0 ns:5 %"))
```

Additional restrictions for this property are:

- Pinpairs should leave the pinpair section empty (`"::"`).

  Example: `"MG2:G:::0 ns:5 %"`

- Match groups the pinpair not reference an explicit pinpair but, if not empty should be general pinpairs (for example, "AD:AR","D:R" etc.)

- If nets and xnets are part of multiple Match Groups, they appear concatenated in the RELATIVE_PROPAGATION_DELAY property. Any pinpairs that are part of the net appear as part of the property at the net or xnet level. This is present to support legacy applications like netlisters. The RPD property that is `dbids` for pinpairs, net and xnets breaks the concatentation. Use `axlMatchGroupProp` for modification of the RELATIVE_PROPAGATION_DELAY property for all objects.

Arguments

t_name	Name of match group (changed to upper case).

Value Returned

`nil`	Error or match group with same name already exists.
o_mgdbid:	`dbid` of match group.

See Also

axlPinPairSeek , axlPinsOfNet , axlMatchGroupCreate, axlMatchGroupDelete, axlMatchGroupAdd, axlMatchGroupRemove, `axlMatchGroupProp`

Example

Create a match group called `MG1`:

```
mg = axlMatchGroupCreate("mg1")
```

axlMatchGroupDelete

```
axlMatchGroupDelete(
                  o_mgdbid/t_mgName
        ) -> t/nil
```

Description

This deletes a match group. The command fails in product tiers that do not support electrical constraints or the symbol editor.

Tip

Using `dbids` is faster than using names.

Arguments

o_mgdbid	`dbid` of a match group.
t_mgName	Name of a match group.

Value Returned

`t`	Match group deleted.
`nil`	Failed.

See Also

`axlMatchGroupCreate`

Examples

Delete match group created in `axlMatchGroupCreate:`

```
mg = car(axlSelectByName("MATCH_GROUP" "MG1"))
axlMatchGroupDelete(mg)
```

or

```
axlMatchGroupDelete("MG1")
```

axlMatchGroupProp

```
axlMatchGroupProp(
                  o_mgdbid/t_mgName
                  o_dbid
                  t_value/nil
         )==> t/nil
```

Description

Adds or removes the RELATIVE_PROPAGATION_DELAY property from a member of a match group. Property must be a legal RPD syntax that includes the RPD name.

The command fails in product tiers that do not support electrical constraints or the symbol editor.

See discussion in `axlDBMatchGroupCreate`.

Tip

Using `dbids` is faster than using names.

Arguments

`o_mgdbid`	`dbid` of a match group
`t_mgName`	Name of a match group
`o_dbid`	Legal database dbid to of a member of the match group
`t_value`	RELATIVE_PROPAGATION_DELAY value in legal syntax. If value is `nil;` removes the property.

Value Returned

`t`	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

o_mgdbid	`dbid` of a match group.
t_mgName	Name of a match group.
o_dbid	Legal database `dbid` to remove from match group.
lo_dbid	List of legal database `dbids` to remove from match group.

Value Returned

`t`	Removed elements.
`nil`	Failed one or more element removals.

See Also

`axlMatchGroupCreate`

Example

To match group in example `axlMatchGroupAdd` remove one of the nets:

```
axlMatchGroupRemove(mg  car(nets))
```

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 un-ordered pins or ratTs on the same net. For example, pinpair `u1.2:r1.2` is the same pinpair as `r1.2:u1.2`. If the pinpair already exists then the existing pinpair is returned. The command fails in product tiers that do not support electrical constraints or the symbol editor.

Note: You cannot create a pinpair if both pins (or ratT) do not belong to the same xnet.

If the pinpair was created in an ECset, the ECsetDerived attribute will be `t` and cannot delete it. You must modify the pinpair in the associated ECset.

At database save, a pinpair must be part of a match group or have a legal electrical constraint property assigned to it. Legal electrical properties are:

- PROPAGATION_DELAY
- MIN_FIRST_SWITCH
- MAX_FINAL_SETTLE
- IMPEDANCE_RULE
- TIMING_DELAY_OVERRIDE
- RELATIVE_SKEW

RELATIVE_PROPAGATION_DELAY is stored on the RPD attrbiute as a list of lists.

See `axlMatchGroupCreate` for more infomation.

Tip

Using `dbids` is faster than using names.

Arguments

o_pin1/o_pin2	dbid of a pin or ratT
t_pin1/t_pin2	A pin name (<refdes>.<pin#>); ratT names are not supported.
o_pinpair	Pinpair dbid.
lo_pinpair	List of pinpair dbids (delete mode only).

Value Returned

Returns depending upon the mode.

nil: error

t	Deletion was successful.
o_pinpair	dbid of added or modified differential pair.

See Also

axlPinPairSeek, axlPinsOfNet, axlMatchGroupCreate

Examples

Example 1: Xnet having two nets; NET1 and NET1A. This demonstrates that pinpairs are stored on the xnet.

Create pinpair using name:

```
pp = axlPinPair("U2.13" "R1.2")
```

Create pinpair with ratT of net NET1A:

```
ratTs = axlPinsOfNet("NET1A" 'ratT)
pp = axlPinPair("U2.13" car(ratTs))
```

Verify pinpairs are both on `NET1A`:

```
n = car(axlSelectByName("NET" "NET1A"))
n->pinpair RETURNS nil
```

Examine the xnet (`NET1`):

```
xn = car(axlSelectByName("XNET" "NET1"))
xn->pinpair RETURNS (dbid:21697512 dbid:21697232)
```

Delete all pinpairs of Example 1:

```
axlPinPair(xn->pinpair)
```

axlPinPairSeek

```
axlPinPairSeek(
                o_pin1
                o_pin2
            )==> o_pinpair/nil
```

Description

Given two pins or ratTs reports if they are part of a pinpair.

Arguments

o_pin1/o_pin2	`dbid` of a pin or ratT.
t_pin1/t_pin2	A pin name (`<refdes>.<pin#>`); ratT names not supported.

Value Returned

o_pinpair	Pinpair `dbid`.
`nil`	Pinpair for given pins does not exist.

See Also

`axlPinPair`

Example

See if a pinpair for these two pins exists:

```
pp = axlPinPair("U2.13" "R1.2")
```

axlPinsOfNet

```
axlPinsOfNet(
                 o_net/t_net
                 g_mode
        ) -> lo_pins/nil
```

Description

Returns list of pins and ratTs on a net or xnet. First argument can be either a net, xnet `dbid` or a net name (xnet names are not supported). Second option, *`g_mode`*, can be *`'pin`* to return only the pins, *`'ratT`* to return list of ratTs or nil to return both pins and ratTs. There is no meaning conveyed in the list of items returned.

Arguments

`o_net`	`dbid` of a net or xnet.
`t_net`	Name of a net (does not support xnet names).
`g_mode`	`nil` return both pins and ratTs of a net.
`'pin`	Return only pins.
`'ratT`	Return only the ratT's.

Value Returned

`lo_pins`	List of pins and/or ratTs on net or xnet.
`nil`	Nothing meeting criteria (or error, `dbid` not net or xnet).

Examples

All pins on GND:

```
net = car(axlSelectByName("NET" "GND"))
lpins = axlPinsOfNet(net, 'pins)
```

All pins and ratTs on first xnet in design root (could be a net):

```
xnet = car( axlDBGetDesign()->xnet )
lpins = axlPinsOfNet(xnet, nil)
```

axlRemoveNet

```
axlRemoveNet(
     t_name
)
⇒ t/nil
axlRemoveNet(
     o_dbid
)
⇒ t/nil
```

Description

Removes a net. May either give a string with the net name to be renamed or *dbid* of an object on that net.

The net name may be used in properties. This function does not update these values.

Arguments

t_name	Net name.
o_dbid	*dbid* of a net.

Value Returned

t	Net successfully removed.
nil	No net is removed.

axlRenameNet

```
axlRenameNet(
     t_old_name
     t_new_name
)
⇒ t/nil

axlRenameNet(
     o_dbid
     t_new_name
)
⇒ t/nil
```

Description

Renames a net. For the old object, may either give a string with the net name to be renamed, or *dbid* of an object on that net. Fails if the new net name already exists in the database.

```
dbid = axlSingleSelectName("NET" '("NET"))
```

Note: This function does not refresh any axl *dbids* to reflect the new net name.

The net name may be used in properties. This function does not update these values.

Arguments

t_old_name	Existing net name.
o_dbid	*dbid* of an object on a net.
t_new_name	New net name (should not exist in the database.)

Value Returned

t	Net successfully renamed.
nil	Net name already exists in the database.

Example

```
axlRenameNet("GND" "NEWGND")

; first verify the new net name doesn't exist
axlSetFindFilter(?enabled '("noall" "nets"))
if(axlSingleSelectName("NET" '("NEWGND") ) then
     axlRenameNet(dbid "NEWGND")
```

axlRenameRefdes

```
axlRenameRefdes(
     t_old_name/o_oldCompDbid
     t_new_name/o_newCompDbid
)
⇒ t/nil
```

Description

Renames a refdes. For either argument, may use a refdes name or a component instance.

If both refdes exist, a swap is done.

Arguments

t_oldName	Existing refdes name.
o_oldCompDbid	Component *dbid*.
t_newName	New refdes name.
o_newCompDbid	Component *dbid*.

Value Returned

t	Refdes successfully renamed or swapped.
nil	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

o_net	*dbid* of net
t_net	Name of net
g_userSchedule(optional)	If t returns the schdule if the net is user schedule or partial userschedule. Other nets in this netin format if nil remove

Values Returned

t_schedule	Schedule or partial user schedule.
nil	Failed or net is not user schedule or partial user schedule. Use axlDebug 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 connected 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

o_net	*dbid* of net
t_net	Name of net
t_schedule	Schedule or partial schedule using $SCHEDULE netin format; if *nil,* remove

Value Returned

t	Schedule applied.
nil	Failed or arguments are incorrect (use `axlDebug`) 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 a .txt file extension. For example, if component definition (compDef) device type is `CAP1`, then the device file name is `cap1.txt`.

Also creates a `devices.map` file which is empty unless the device name has characters that are not legal as a filename.

See `netin` documentation for device file syntax.

Note: Do not use this function if you use Cadence Front-End Schematic packages.

This function overwrites existing <`device`> and `devices.map` files in the directory.

Arguments

`o_compDefDbid`	Component definition of the device file to write out.
`t_output_dir`	Directory in which to write the files. If not provided, the current directory is used.

Value Returned

`t`	Device file successfully written.
`nil`	Failed to write device file due to incorrect *`dbid`* 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

o_symDefDbid	Symbol definition to store on disk.
t_output_dir	Directory in which to store the files. If not provided, the current directory is used.

Value Returned

`t`	Files successfully written.
`nil`	Failed to write files due to incorrect *dbid* 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.

A

Building Contexts in Allegro

Introduction

A context via can be created by either of two methods – standard and autoload. Both methods substantially improve performance of Skill code loading. Even more benefits can accrue if you combine several Skill files into one context. The autoload method is a super-set of standard contexts and offers deferred context loading functionality. The autoload method is used by all Allegro provided contexts.

The standard contexts are much easier to build, are more evident to the user, and typically require more memory. The autoload contexts are much harder to build, but the system only loads the contexts upon demand. For a more complete discussion of the differences, see the section on Contexts in the *Skill Language User Guide*.

Requirements

You must have a Skill developers license and the `il_allegro` program. Currently, this license is only available on UNIX. The `il_allegro` program is part of every standard Allegro release.

Cautions

Most Skill code can be built into contexts. However, there are several potential problems that you should keep in mind when writing code. A complete discussion of these issues can be found in Chapter 10 of the *Skill Language User Guide*.

Note: Cadence recommends that you prefix your Skill functions with upper case prefixes. This minimizes the chance of naming collisions with Cadence Skill functions that use lower case prefixes.

Additionally, autoload contexts have some additional cautions. Please adhere to the following guidelines:

Autoload Context Guidelines

- Files put into an autoload context should only contain variables and procedures (functions).
- Do not load other skill files. Have `startup.il` load them.
- Do not call `axlCmdRegister`.
- Do not do anything outside of a procedure – it will not work.

Building Standard Contexts

To build a standard context

1. Create a directory that has all the Skill files to be built into the context.
2. Add the `startup.il` file (see File B1 on page 1216).
3. Create a Skill function with the same name as your context that registers your commands with the Allegro shell. This step is required in allegro_designer if you wish to access your Skill code. Only one of these functions is permitted per context. The function name must be the same as the context name. This step is analogous to the `.ini` file in autoload contexts.

   Format:

   ```
   (defun <ContextName> ()
   (axlCmdRegister "mycommand" '<MYSkillCommand> ?cmdType ....)
   .... other axlCmdRegister ..
   )
   ```

   Example:

   ```
   (defun MYTEST()
   (axlCmdRegister "mytest" 'MYTest ?cmdType "general")
   )
   ```

4. Run the `buildcxt <ContextName>` script (see File S1 on page 1216). This produces a single file named, `<ContextName>.cxt`. For example: `buildcxt MYTEST`.

   To load the context into allegro_designer, issue the Allegro command `loadcontext <ContextName>`. In programs where the Skill type-in mode is available, the Skill functions `loadContext <contextName.cxt>` and `callInitProc <ContextName>` perform the same function.

   Example:

```
Allegro > loadcontext MYTEST
```

Skill version:

```
skill > (loadContext "MYTEST.txt")
skill > (callInitProc "MYTEST")
```

Building Autoload Contexts

To build a context by the autoload method

1. Create directory hierarchies:

   `./pvt/etc/context`

   `./etc/context`

2. Under `./pvt/etc/context`, create a directory using your context name and populate it with your Skill files.

3. Add a `startup.il` (see File B1 on page 1216) to the mix and stir well.

4. Insure that the `cxtFuncs.il` (see File A1 on page 1217) is in the root directory.

5. Run the `buildautocxt` (see File A2 on page 1220) UNIX command with your context name. For example: `buildautocxt <myContext>`.

6. If the context build is successful, you will have 3 files in the `./etc/context` directory with your context name (`.aux, .cxt, .toc`).

7. Add an optional fourth file with a `<myContext>.ini` that has your `axlCmdRegister`. If you do not wish to register your Skill commands as Allegro commands, you may skip this step. However, in allegro_designer this is the only method for accessing your Skill code.

   Example:

   ```
   (axlCmdRegister "my_command" 'MYSkillFunction ?cmdType "interactive")
   ```

8. Take the four context files and add them to the directory `<cds_root>/share/pcb/etc/context`.

9. Edit the `pcd` file in this directory for the product requiring the context. Names are:

`allegro.pcd`	All allegro_layout (CBD) based products.
`designer.pcd`	allegro_designer
`apd.pcd`	advanced_package_designer
`floorplan.pcd`	allegro_si

You need to add a line at the end of the file in the following format:

```
<NAME><VERSION><CONTEXTS>
```

Note: Neither the NAME of VERSION is important. It is only used with the Skill function `printBlend`.

Example:

```
MYCONTEXT 1.0MyContext
```

Two environment Bourne variables help in debugging problems in this area. They are:

`CDS_DEBUG_CONTEXTS`	file `/tmp/context.log` - context stats
`CDS_DEBUG_CXTINIT`	file `/tmp/initCxt.log` - context init - also stderr init context print

Files with This Package

File B1

Helper Skill code to load all Skill files in a directory.

```
;-------------------------------------------------------------
;startup.il
foreach(file rexMatchList(".*\\.il$" getDirFiles("."))
    ; don't load myself -- bad idea
    when( nequal(file, "startup.il")
        load(file)
    )
)
;-------------------------------------------------------------
```

File S1

`buildcxt csh` script to for building standard contexts.

```
#!/bin/csh -f
# This builds a standard context see README.cxt for other set-up requirements

if ($#argv != 1) then
   echo "Usage: $0 <context name>"
   echo "Assumes that a startup.il file exists in current directory"
   echo " this file is used to specify the loading of other skill files"
```

```
   exit 1
endif
set theContext = $argv[1]

if (!(-e startup.il)) then
   echo "ERROR: Can't find standard.il file"
   exit 1
endif


il_allegro  << EOF
(setSkillPath ".")
(setContext "$theContext")
(load "startup.il")
(defInitProc "$theContext" '${theContext})
(saveContext "$theContext.cxt")
(exit)
EOF
/bin/rm -f AUTOSAVE.brd

echo ""
echo ""
echo ""
echo "Context will be found $theContext.cxt"
echo ""
exit 0
```

File A1

`cxtFuncs.il` Skill helper program to build autoload contexts.

```
;(
;-----------------------------------------------------
; EXPORTED FUNCTIONS:
;   buildContext : used to build a context
;   getContext   : used to load a context
;
;    Mods -- fxf 8/25/95 to support local building of contextes
;-----------------------------------------------------
;
; Constants
;    ilcDftSourceFileDir : directory name where Skill source
;            files reside
;    ilcDftDeliveryDir   : directory name where delivered
;                          context files are saved.
;    (fxf) may be overridden before calling
;-----------------------------------------------------

unless(boundp('ilcDftSourceFileDir) ilcDftSourceFileDir = "pvt/etc/context")
```

```
unless(boundp('ilcDftDeliveryDir) ilcDftDeliveryDir = "etc/context")

(defun _parsePath (path)
(let (lpath)
        (cond (path
          lpath = parseString(path "/")
          (while (!rindex(car(lpath) "tools")) lpath = cdr(lpath))
          buildString(lpath "/")
          )
          (t nil))
))


_stacktrace = 10
setSkillPath(strcat(". ~ " prependInstallPath("local")))
(cond ((getd 'dbSetPath) (dbSetPath ". ~")))

;
; loadCxt --
;    Load a context and call its init function.
;
(defun loadCxt (cxt cxtPath)
  (let ((f (strcat (cdsGetInstPath cxtPath) "/" cxt ".cxt")))
     (cond
     ((null (isFile f)) nil)
        ((null (loadContext f))
         (printf "load of context %s failed\n" cxt))
     ((null (callInitProc cxt))
         (printf "init proc of context %s failed\n" cxt))
     (t (printf "Loading context %s\n" cxt))
     )
  )
)

;
; buildContext --
;    Build a new context, even if one exists.
;

(defun buildContext (cxt @rest targs)
   (let (cxtPath srcPath fullCxtPath)

     cxtPath = ilcDftDeliveryDir
        (setq srcPath (strcat ilcDftSourceFileDir "/" cxt))

     ;; <fxf>: doesn't allow local contextes so use above 2 lines
        ;;(cond ((car targs) (setq cxtPath (car targs)))
           ;;((setq cxtPath (_parsePath (_iliGetActualCxtPath cxt))) t)
              ;;(t (setq cxtPath ilcDftDeliveryDir)))
        ;;(cond ((cadr targs) (setq srcPath (cadr targs)))
           ;;((setq srcPath (_parsePath (_iliGetActualSrcPath cxt))) t)
```

```
             ;;(t (setq srcPath (strcat ilcDftSourceFileDir "/" cxt))))
        fullCxtPath = cdsGetInstPath(cxtPath)

        (deleteFile (strcat fullCxtPath "/" cxt ".cxt"))
        (deleteFile (strcat fullCxtPath "/" cxt ".al"))
        (deleteFile (strcat fullCxtPath "/" cxt ".ini"))

        (updateContext cxt cxtPath srcPath)
        (updateAutoloads cxt cxtPath srcPath)
))

;
; updateContext --
;    If there is source and it is newer than the context,
;    then build a new context.  Otherwise if there is no source
;    use the existing context.
;
(defun updateContext (cxt cxtPath srcPath)
     (cond ((isDir (cdsGetInstPath srcPath)) (makeCapContext cxt cxtPath srcPath))
        ((loadCxt cxt cxtPath) t)
        (t (printf "Can't find context %s\n" cxt )))
)

(defun updateAutoloads (cxt cxtPath srcPath)
   (let ((afile (sprintf nil "%s/%s.al" (cdsGetInstPath srcPath) cxt))
      (ifile (sprintf nil "%s/%s.ini" (cdsGetInstPath srcPath) cxt)))

     (cond ((isFile ifile) (system (sprintf nil "cp %s %s" ifile (cdsGetInstPath cxtPath))))
           ((isFile afile) (system (sprintf nil "cp %s %s" afile (cdsGetInstPath cxtPath))))
        (t t))
))

;
; getContext --
;    Load the context if it exists, otherwise build it.
;

(defun getContext (cxt @rest targs)
   (let (cxtPath srcPath)
        (cond ((car targs) (setq cxtPath (car targs)))
           ((setq cxtPath (_parsePath (_iliGetActualCxtPath cxt))) t)
              (t (setq cxtPath ilcDftDeliveryDir)))
        (cond ((cadr targs) (setq srcPath (cadr targs)))
           ((setq srcPath (_parsePath (_iliGetActualSrcPath cxt))) t)
              (t (setq srcPath (strcat ilcDftSourceFileDir "/" cxt))))

        (cond ((loadCxt cxt cxtPath) t)
           ((isDir cxt (cdsGetInstPath srcPath))
                (makeCapContext cxt cxtPath srcPath))
         (t (printf "Can't get context %s\n" cxt)
         ))
```

```
))

(sstatus trapDefs ilcDftDeliveryDir)
(sstatus lazyComp nil)
```

File A2

`buildautocxt csh` script to build autoload contexts.

```
#!/bin/csh -f
# This builds a context see README.cxt for other set-up requirements

if ($#argv != 1) then
   echo "Usage: $0 <context name>"
   exit 1
endif
set theContext = $argv[1]

if (!(-e pvt/etc/context/$argv[1])) then
    echo "pvt/etc/context/$argv[1] does not exit"
    exit 1
endif

if (!(-e etc/context)) then
    mkdir -p etc/context
endif


il_allegro -ilLoadIL cxtFuncs.il << EOF
(getContext "skillCore")
(setSkillPath ".")
(cdsSetInstPath ".")
buildContext "$theContext"
exit
EOF

echo ""
echo ""
echo ""
echo "Context files will be found at etc/context/$theContext.*"
echo ""
exit 0
```