Symphony SKILL Development Guide

Product Version 23.1
September 2023

Contents

1

Symphony Behavior Overview

To protect data integrity, Symphony controls the flow of incoming and outgoing database updates using commands and database transactions.

Command Mechanism

Symphony uses the command mechanism of layout editors to enable Symphony functionality as well as to prevent database conflicts from changes made by other clients.

This guide describes features and functionality for the following layout editors:

- Allegro PCB Designer
- Allegro Package Designer L (using SiP Layout XL)
- SiP Layout XL

Note: The information provided in this document is based on Cadence® Allegro® release 17.2-2016 hotfix 0038 (QIR6).

Functionality Control

A command must be explicitly marked as Symphony-enabled to transmit database changes to the Symphony server. If a command is not Symphony-enabled, no database changes made during the command will be sent to the server, thus causing design becoming out of sync with the server design. Only commands that make supported database changes should be Symphony-enabled. A command does not need to be Symphony-enabled if it does not make changes to the database.

Update Blocking

When a command is active, incoming updates are prevented from being applied to the database, allowing the command to access the database without having to worry about conflicts caused by other clients. This is true for any layout editor command. As an example, if a database object is deleted by another client while being operated on in the local client, the local operation would fail. All incoming updates are queued until the command is finished, at which time they are applied to the database.

SKILL Restriction

As commands are required to control Symphony behavior, SKILL code must be run as an layout editor command to be supported in the Symphony environment. Database changes made outside of a command are not sent to the Symphony server, and therefore cause the design to become out of sync with the server design. Additionally, SKILL code running outside of a command may be interrupted by incoming updates from other clients. For more information on Allegro SKILL commands., see *Chapter 20: Command Control Functions* of the *Allegro User Guide: SKILL Reference*.

Transactions

Supported database changes made during a Symphony-enabled command are tracked by the transaction system and are automatically broadcast to the server when the main transaction is committed to the database. Changes made outside of a transaction are not broadcast to the Symphony server and therefore cause the design to become out of sync with the server design.

For more information about transactions, see *Chapter 18: Database Transaction Functions* of the *Allegro User Guide: SKILL Reference*.

Additionally, intermediate design changes are broadcast to the server when specialized Symphony transactions are committed to the database. These “temp updates” perform several functions. First, these updates are used to display dynamic graphics on other clients showing changes that have been made, but not yet committed to the database. This is assists other clients from avoiding conflicts. Second, multi-user locks will be generated for all database objects that are included in the temp update. This will prevent other clients from making changes to these objects until the main transaction is committed or the command is canceled.

Note: Many, but not all, low-level Allegro SKILL functions create transactions for database changes. This meets the transaction requirement and causes the change to be broadcasted to the Symphony server. However, this may result in the broadcast of many small changes, which could have a negative performance impact. It is a recommended practice to wrap groups of changes in transactions at the command level.

Rejections

The Symphony server holds the master copy of the database and is responsible for maintaining its integrity. As clients send database updates, the server evaluates the changes and rejects any update that causes conflicts. If an update is rejected by the server, it is discarded and a rejection notification is sent to the originating client. The client automatically attempts to back out the rejected update and continue with normal operation. If the client is unable to back out the change, the client is forced to reload the design from the server.

Locks

Multi-user locks are used to help minimize server rejections by preventing multiple clients from modifying the same objects at the same time. When an object has a multi-user lock, it behaves as a fixed object for all clients except the creator of the lock. There are three types of locks:

- Permanent Locks – These are locks that are explicitly added to an database object by a client. The locks remain until they are explicitly removed, the client disconnects from the Symphony session, or the Symphony session ends. The locks can be explicitly removed by the client that created them, or from the Symphony Server UI.
- Command Locks – These are locks that are generated during a command run. These can be requested explicitly from the command, or are generated automatically by temporary database updates. These locks last until they are explicitly removed by the command or until the command is completed.
- Local Locks – These locks are generated by an incoming database update that has not yet been applied to the local database. For example, while a client is in an command, all incoming updates are queued until the command is completed. The database objects contained in the queued updates are automatically locked to prevent the client from modifying them in the command. These locks remain until the update is applied to the client’s database.

Although not necessary, it is generally beneficial for commands to request locks for objects upon which they are operating. By locking the objects as early as possible, the likelihood of another client modifying those objects at the same time is reduced.

Allegro SKILL Support

Symphony supports a subset of Allegro SKILL functions. Any unsupported database operations that are performed in a SKILL command are not be sent to the Symphony server and therefore cause the design to become out of sync with the server design. Additionally, there are Symphony specific restrictions on supported operations.

Caution

There may be no immediate indication that the local database is out of sync with the server.

Unsupported changes made to the local database are not available in the server version. This can cause subsequent changes on the client to be rejected by the server. If the local database is out of sync, reloading the master database from the server resolves the issue.

Supported Allegro SKILL Functions

Following SKILL functionality is supported in the Symphony environment:

- Color/Visibility – Changes to color/visibility only apply to the local client and are not transmitted to the Symphony server. Some of these changes, such as layer color and visibility, are maintained between sessions for a user ID. Other changes, such as custom colors, are not currently maintained and are reset between sessions.
- Selection and Find
- UI Functionality – This includes interface functions such as `axlEnterPoint`, menu related functions such as `axlUIMenuLoad`, and form related functions such as `axlFormCreate`.
- Geometry/Shape Changes
- Database Read Functions – This includes functions that query the database without making changes such as most `axlGet*` functions.

See Appendix 3, “AXL Function Support.” for the full list of SKILL functions and their status in Symphony.

Unsupported Allegro SKILL Functions

Generally, the following SKILL functionality is not supported in the Symphony environment. If these changes are performed in a Symphony session, they are not broadcast to the server and therefore cause the design to become out of sync with the server design. Unsupported SKILL functions are:

- Layer/Cross-Section Changes
- Text Block Changes
- Constraint Changes
- Symbol/Component Changes
- Padstack Changes
- Net/Logic Changes
- Property Changes
- Database Attachment Changes

See Appendix 3, “AXL Function Support,” for the full list of SKILL functions and their status in Symphony.

Unsupported Layers

Changes to some design layers are not supported in the Symphony environment. Changes to these layers are not broadcast to the server and therefore cause the design to become out of sync with the server design. The unsupported layers are:

- All subclasses on the “Plan” class.
- All subclasses on the “Drc Error Class” class.
- Manufacturing/Autosilk_Top
- Manufacturing/Autosilk_Bottom

To test whether a layer is supported, the Symphony SKILL API function axlMUIsLayerSupported can be called.

Unsupported Objects

Changes to some objects are not supported in the Symphony environment. Similarly, certain objects are supported for some operations but not others. For example, changes to dimension objects are not supported, and symbols can be moved, but not deleted.

To determine if a given operation is supported on a given object type, the Symphony SKILL API function axlMUIsObjectSupported can be called.

Identifying Unsupported Functions

To identify unsupported SKILL functions, a new rule has been added to the `sklint` utility. The new rule is enabled under one or more of the following conditions:

- The `sklint` utility is run while connected to a Symphony session.
- The environment variable `MU_SKLINT_ENABLE` is set. This takes effect immediately and can be set from the command line window of the layout editor.

If an unsupported SKILL function is encountered, `sklint` throws a warning of the following form:

```
WARN (MUCOMPAT): test.il, line 35 (testMainFunc) : Function axlCNSDelete not
supported in multi-user mode.
```

For information about `sklint`, see *Chapter 3: Lint Functions* of the *Cadence SKILL Developer Reference*.

Symphony SKILL API

A number of new API functions are provided to improve Symphony support for new and existing SKILL code. All multi-user related API functions will begin with the “axlMU” prefix.

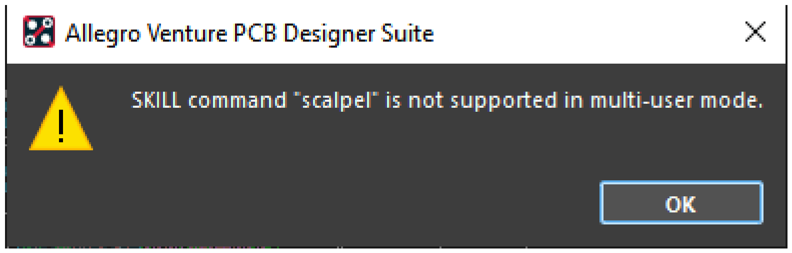
See Appendix A for complete descriptions of the API functions.

Writing Symphony Enabled SKILL

Writing commands for use in the Symphony environment is not fundamentally different from writing commands in the single-user environment. (See the “Allegro User Guide: SKILL Reference” documentation for information on writing commands in SKILL.) The main exception is the command must be explicitly enabled in the Symphony environment.

Enabling SKILL Commands in Symphony

By default, all SKILL commands are disabled in a Symphony session. If a SKILL command is run in the default setting, the following message is displayed.



SKILL command "scalpel" is not supported in multi-user mode.

Figure 1-1 Command Not Supported Dialog

SKILL commands are enabled by adding them to the `symphony_skill.txt` file located in the `pcbenv` directory at user installation or at the site location, specified by the `allegro_site` environment variable. By default, all commands are enabled in ‘read-only’ mode. This means that changes made by the command are not broadcast to the Symphony server.

To enable broadcasting of database changes, the `rw` keyword must be added to indicate that this is a ‘read-write’ command:

```
"testCommandReadOnly"      ; This is a read-only command.  No changes will
                           ; be broadcast to the Symphony server.

"testCommandReadWrite" rw  ; This is a read-write command.  Supported
                           ; database changes will be broadcast to the
                           ; Symphony server.
```

Note: It not necessary to specify a command as “rw” if it does not make supported database changes.

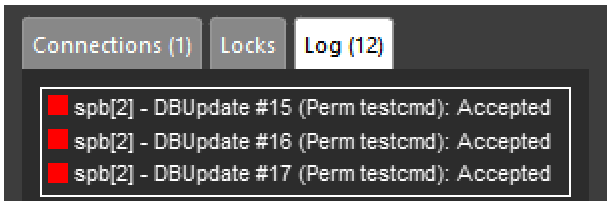
Transaction Example – Grouping Updates

All supported database changes must be performed inside of a database transaction. Supported changes are broadcast when the outer transaction is committed. Consider the following example:

Skill Code

```
; Register the command.
(axlCmdRegister "testCmd" '_testCmdMain ?cmdType "interactive")

; Main function for “testCmd”.
(defun _testCmdMain ()
; Add three lines on ETCH/TOP
(axlDBCreateLine (list 0.0:0.0 100.0:0.0) 10.0 "ETCH/TOP")
(axlDBCreateLine (list 0.0:100.0 100.0:100.0) 10.0 "ETCH/TOP")
(axlDBCreateLine (list 0.0:200.0 100.0:200.0) 10.0 "ETCH/TOP")
)
```

Server Log



```
spb[2] - DBUpdate #15 (Perm testcmd): Accepted
spb[2] - DBUpdate #16 (Perm testcmd): Accepted
spb[2] - DBUpdate #17 (Perm testcmd): Accepted
```

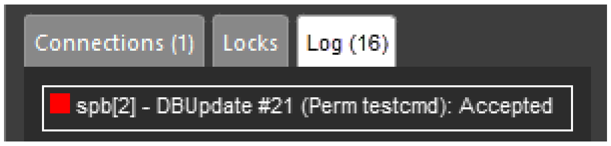
Figure 1-2 Transaction Example - No Transaction Added

Because the `axlDBCreateLine` function generates its own transaction, each change broadcasts to the server separately. Wrapping the entire function in a transaction, reduces this to a single update.

Skill Code

```
; Register the command.
(axlCmdRegister "testCmd" '_testCmdMain ?cmdType "interactive")

; Main function for “testCmd”.
(defun _testCmdMain ()
(let (transMark)
; Start transaction.
(setq transMark (axlDBTransactionStart))

; Add three lines on ETCH/TOP
(axlDBCreateLine (list 0.0:0.0 100.0:0.0) 10.0 "ETCH/TOP")
(axlDBCreateLine (list 0.0:100.0 100.0:100.0) 10.0 "ETCH/TOP")
(axlDBCreateLine (list 0.0:200.0 100.0:200.0) 10.0 "ETCH/TOP")

; Commit transaction.
(axlDBTransactionCommit transMark)
))
```

Server Log

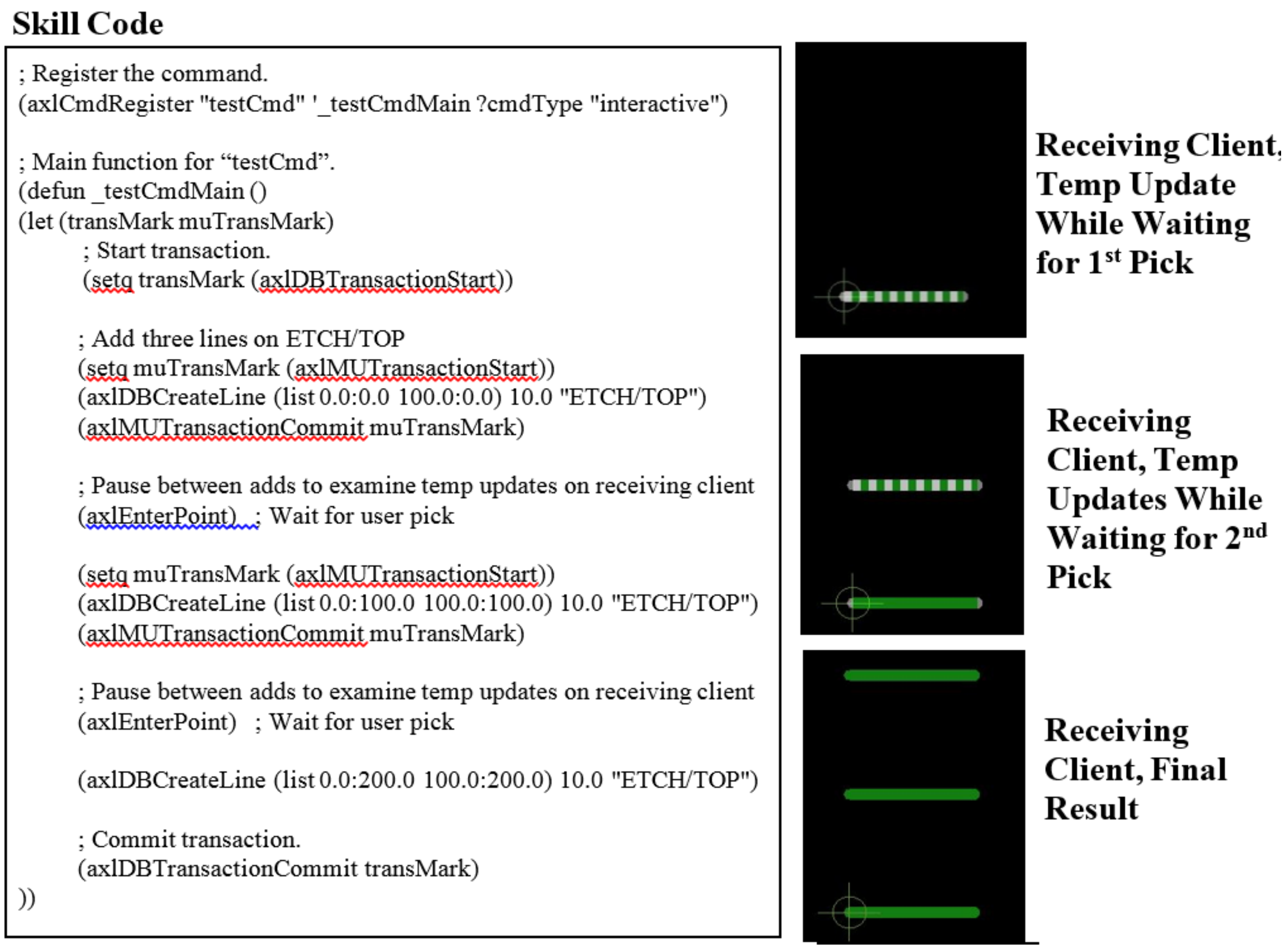


```
spb[2] - DBUpdate #21 (Perm testcmd): Accepted
```

Figure 1-3 Transaction Example - Transaction Added

Transaction Example – Broadcasting Temp Updates

In order to send intermediate update information to the server, and generate temporary graphics and automatic locks during a command, changes can be wrapped in transactions using the Symphony SKILL API functions axlMUTransactionStart and axlMUTransactionCommit.



Skill Code

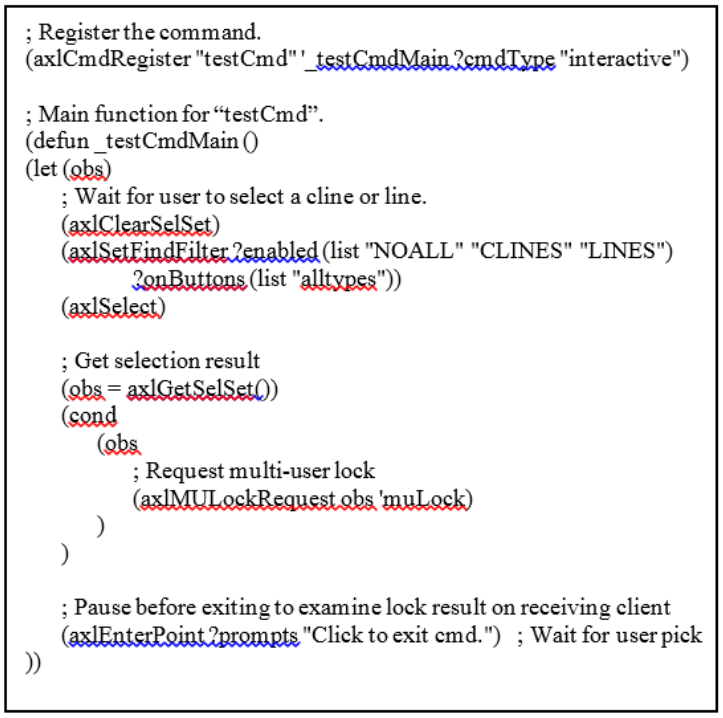
```
; Register the command.
(axlCmdRegister "testCmd" '_testCmdMain ?cmdType "interactive")

; Main function for "testCmd".
(defun _testCmdMain ()
(let (transMark muTransMark)
        ; Start transaction.
        (setq transMark (axlDBTransactionStart))

        ; Add three lines on ETCH/TOP
        (setq muTransMark (axlMUTransactionStart))
        (axlDBCreateLine (list 0.0:0.0 100.0:0.0) 10.0 "ETCH/TOP")
        (axlMUTransactionCommit muTransMark)

        ; Pause between adds to examine temp updates on receiving client
        (axlEnterPoint)  ; Wait for user pick

        (setq muTransMark (axlMUTransactionStart))
        (axlDBCreateLine (list 0.0:100.0 100.0:100.0) 10.0 "ETCH/TOP")
        (axlMUTransactionCommit muTransMark)

        ; Pause between adds to examine temp updates on receiving client
        (axlEnterPoint)  ; Wait for user pick

        (axlDBCreateLine (list 0.0:200.0 100.0:200.0) 10.0 "ETCH/TOP")

        ; Commit transaction.
        (axlDBTransactionCommit transMark)
))
```

Receiving Client, Temp Update While Waiting for 1st Pick
Receiving Client, Temp Updates While Waiting for 2nd Pick
Receiving Client, Final Result

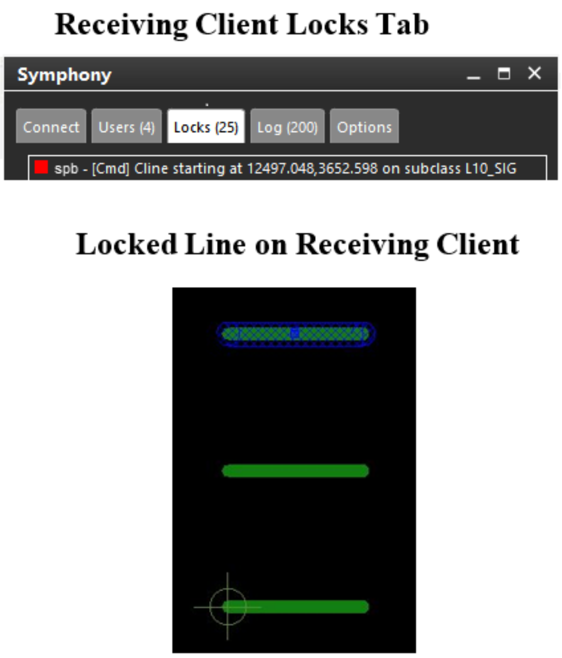
Figure 1-4 Transaction Example - Temp Updates

Requesting Locks

When operating on database objects, multi-user locks can be requested to prevent other clients from modifying those objects. The code shown in the following figure waits for the user to select a line or cline and requests a multi-user lock for the selected object.



```
; Register the command.
(axlCmdRegister "testCmd" '_testCmdMain ?cmdType "interactive")

; Main function for "testCmd".
(defun _testCmdMain ()
(let (obs)
    ; Wait for user to select a cline or line.
    (axlClearSelSet)
    (axlSetFindFilter ?enabled (list "NOALL" "CLINES" "LINES")
          ?onButtons (list "alltypes"))
    (axlSelect)

    ; Get selection result
    (obs = axlGetSelSet())
    (cond
        (obs
          ; Request multi-user lock
          (axlMULockRequest obs 'muLock)
        )
    )

    ; Pause before exiting to examine lock result on receiving client
    (axlEnterPoint ?prompts "Click to exit cmd.")  ; Wait for user pick
))
```



spb - [Cmd] Cline starting at 12497.048,3652.598 on subclass L10_SIG

Figure 1-5 Requesting Locks

Executing SKILL Outside of a Command

Although not recommended, it is possible to safely execute SKILL code outside of a command in the Symphony environment only if the following conditions are met:

- The SKILL code must only perform read-only operations. No database changes may be made.
- Processing of incoming database updates must be disabled using the axlMUSetUpdateProcessingPause API function. This allows the running SKILL code to access the database without changes to the database being made by other clients. Database updates must be re-enabled after the SKILL code is finished.

2

SKILL API

All Symphony SKILL API functions begin with the ‘axlMU’, prefix for ‘axl Multi-User’.

axlMUIsMultiUser

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

Description

Determines if the Allegro PCB Editor is running in multi-user mode.

If Allegro PCB Editor is currently connected to a multi-user server, the value ‘t’ is returned.

Arguments

None

Value Returned

`t`	In multi-user mode
`nil`	In single-user mode.

axlMUIsLayerSupported

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

Description

Determine if the specified layer is supported in the multi-user mode. If the layer is not supported, changes made on that layer are not be passed to the multi-user server and cannot be seen by other clients. Commands should not allow changes on unsupported layers in order to be compatible with multi-user mode.

Arguments

t_layer	layer name For example: "ETCH/TOP", "MANUFACTURING"

Value Returned

t - Layer is supported.

nil - Layer is not supported.

Note: If no slash in the given layer name, assume just a class is given.

See Also

axlMUIsMultiUser

axlMUIsObjectSupported

```
axlMUIsObjectSupported(
     o_dbid/lo_dbid
     [g_mode]
     )
     ==> t/nil
```

Description

Determines if changes to the given allegro database object or objects are supported in the multi-user mode. If changes are not supported, changes made to that object are not passed to the multi-user server and are therefore not seen by other clients.

For a given object, some operations may be supported while others may not. For example, symbols can be moved, but cannot be deleted.

Arguments

`o_dbid/ lo_dbid`	Single AXL DB ID or list of DB IDs. For a list of objects, if the result is nil for any object, the function will return nil
`g_mode`	Optional mode argument. This describes the type of action being performed. Possible `g_mode` values are: ❑ 'muCopy: Can the object be copied. ❑ 'muCreate: Can the object be created. ❑ 'muDelete - Can the object be deleted. ❑ 'muMove - Can the object be moved. ❑ 'muModify - Can the object be modified. This includes any physical change not covered by the other modes, such as changing line width. When the `g_mode` is not specified, a return value of `t` indicates that at least one action is supported for the specified object

Value Returned

`t`	Object can be modified.
`nil`	Object can not be modified.

See Also

axlMUIsMultiUser, axlMUIsLayerSupported

axlMUTransactionStart

```
axlMUTransactionStart(
     )
     ==> x_mark/nil
```

Description

Starts a database transaction used to group multi-user changes into a single Temp update. Temp updates are the ‘dynamic’ display items that are shown to other users connected in a multi-user session when database changes are made. These updates are driven by the transaction system and are generated each time the transaction started by this function is committed. If nested calls are made to this function, only the outermost transaction will generate a Temp update.

Additionally, this function controls the temporary (temp) updates sent to the mutli-user server. This is used with other `axlMUTransaction` functions.

Note: This function is similar to axlDBTransactionStart in that it is used to mark the start of a transaction to the database. For more information, see axlDBTransactionStart documentation.

The transaction mark returned by `axlMUTransactionStart` must be passed to the `axlMUTransactionCommit` or `axlMUTransactionRollback` functions.

In single-user mode, the function behavior is same as calling `axlDBTransactionStart` function without arguments.

Arguments

None

Value Returned

- nil if not successful
- An integer mark if started transaction.

Note: Large numbers of Temp updates may produce performance issues. Care should be taken when determining when it is appropriate to send a temp update to the server.

See Also

axlMUIsMultiUser, axlMUTransactionCommit, axlMUTransactionRollback, axlDBTransactionStart

axlMUTransactionCommit

```
axlMUTransactionCommit(
     x_mark
     )
     ==> t/nil
```

Description

Commits a database transaction used to group multi-user changes into a single Temp update. Additionally, the transaction committed by this function groups all changes into a single Temp update to be transmitted to the multi-user server.

This is used with other `axlMUTransaction` functions. The transaction mark passed to `axlMUTransactionCommit` must be created by the `axlMUTransactionStart` function.

This function is similar to `axlDBTransactionCommit` in that it is used to commit a database transaction to the database (For more information, see the documentation for `axlDBTransactionCommit`). In single-user mode, this function behaves the same as the `axlDBTransactionCommit` function.

Arguments

x_mark	Database transaction mark returned by the axlMUTransactionStart function

Value Returned

- `nil` if function call is not successful
- `t` if successful

See Also

axlMUIsMultiUser, axlMUTransactionStart, axlMUTransactionRollback, axlDBTransactionCommit

axlMUTransactionRollback

```
axlMUTransactionRollback(
     x_mark
     )
     ==> t/nil
```

Description

Rollback a database transaction used to group multi-user changes into a single temp update. This function is similar to `axlDBTransactionRollback` in that it is used to undo a database transaction (see the documentation for `axlDBTransactionRollback` for more information).

This is used with other `axlMUTransaction` functions.The transaction mark passed to `axlMUTransactionRollback` must be created by the `axlMUTransactionStart` function.

Arguments

x_mark	database transaction mark returned by `axlMUTransactionStart`

Value Returned

`nil` if not successful, `t` if successful

Note: In single-user mode, this function behaves the same as the `axlDBTransactionRollback` function.

See Also

axlMUIsMultiUser, axlMUTransactionStart, axlMUTransactionCommit, axlDBTransactionRollback

axlMUSetUpdateProcessingPause

```
axlMUSetUpdateProcessingPause(
     g_value
     )
     ==> t/nil
```

Description

Disables or enables processing of incoming multi-user database updates. If processing is paused, incoming updates will continue to be received and will be queued up until the processing is re-enabled. This function should be used to temporarily suspend update processing in order to examine Allegro database objects in multi-user mode when not running inside of a command. (Processing of updates is automatically paused when running inside of a command.) If update processing is not paused, then any database objects being examined may be changed or deleted by another user while still in use which may cause instability.

The previous value of update processing is returned and should be passed back to the function when update processing may be resumed. This allows nested calls to this function.

Arguments

g_value - `t` or `nil` to indicate if multi-user update processing should be paused or resumed.

Value Returned

`t/nil` - Returns the previous value of the pause setting.

Important

This function only pauses processing of permanent multi-user updates, which are the updates that make changes to the database. Temporary updates, which are used only to dynamically display changes being made by other users, are not affected by this setting and will continue to be processed.

This function has no effect in the single-user environment.

See Also

axlMUIsMultiUser

Example (pseudo code)

; We are not in a command and want to examine the db, pause incoming multi-user updates.

```
prevValue = axlMUSetUpdateProcessingPause(t)
... do work ...
```

; We are all done with our operations, restore multi-user updates to the previous setting.

```
    axlMUSetUpdateProcessingPause(prevValue)
```

axlMULockRequest

```
axlMULockRequest(
     o_dbid/lo_dbid
     g_mode
     [g_type]
     )
     ==> t/nil
```

Description

This function requests that the given Allegro database object or list of objects be locked or unlocked in the multi-user environment.

Arguments

`o_dbid/lo_dbid`	Single AXL DB ID or list of DB ID's
`g_mode`	`'muLock` or `'muUnlock`
`g_type`	Optional argument indicating type of lock being requested. Either command or permanent locks can be created. If this argument is not specified, the default is command locks. Possible values: ❑ 'muLockTypeCmd - Command lock. ❑ 'muLockTypePerm - Permanent lock.

Command locks are automatically removed when the current command ends. Permanent locks remain until explicitly removed.

Value Returned

`t/nil`	Returns `t` if the lock request was sent successfully. Note that this does not indicate whether or not the request was accepted or rejected by the server. This function returns immediately after the request is sent without waiting for a reply.

If a command lock request is rejected while the command is still active, the default behavior is to cancel the current command. No action is performed if an command unlock request is rejected. This does not apply to permanent locks.

Note: Requesting multi-user locks is recommended, but not required. If an Allegro object is not locked while being operated on, there is a greater chance that another user may modify the object at the same time. This will result in the operation being rejected by the multi-user server.

All command locks are automatically removed at the end of a command.

This function has no effect in the single-user environment.

See Also

axlMUIsMultiUser

axlMUGetLocks

```
axlMUGetLocks(
     )
     ==> l_lockList/nil
```

Description

This function returns a list of Allegro database objects that are currently multi-user locked.

Arguments

Nothing

Value Returned

l_lockList	List of currently locked objects or nil if not locks present. The returned list will be a list of lists with the following format.
	`((o_dbid l_userId g_ownedByMe (l_locktypes))`
	`(o_dbid l_userId g_ownedByMe (l_locktypes))`
	`...`
	`)`

o_dbid	AXL DB ID of locked object
l_userId	User ID of lock owner. Format is same as returned by axlMUGetUserId.
g_ownedByMe	t if this lock owned by current user, nil if not
l_lockType	list of lock types that apply to this object. Valid values are: ■ `'muLockTypeCmd` ■ `'muLockTypePerm` ■ `'muLockTypeLocal`

Locks of type 'local are automatically generated for incoming database updates that have not yet been applied to the database. These locks prevent objects that are changed in the update from getting modified before the update can be applied.

See Also

axlMUIsMultiUser, axlMULockRequest, axlMUGetUserId

axlMUGetUserId

```
axlMUGetUserId(
     )
     ==> l_userId/nil
```

Description

This function returns the multi-user user id for the current user. In the single-user mode, the return value is `nil`. Format of user id is:

```
(t_userName n_userIdNumber)
```

where

`t_userName` – represents the login name

`n_userIdNumber` – Represents the ID number unique to each multi-user session.This value may be different each time a session is joined.

Arguments

None

Value Returned

`t_userId`	User ID
	`nil` if not connected to a multi-user server

Note: The user ID string may change between multi-user sessions. If the server disconnects, the current user ID will no longer be valid.

See Also

axlMUIsMultiUser

3 AXL Function Support

Following table provides a complete list of the public AXL API functions. Each is categorized based on how it interacts with Symphony. Unsupported functions may not be used during a Symphony session without causing the design to become out of sync with the server.

Function Category

R = ReadOnly (Supported) - Does not make db changes

S = Supported - Makes supported db changes

I = Changes Ignored (Supported) - Makes db changes that are ignored by Symphony

U = Unsupported - Makes db changes that are not sent to server

B = Obsolete - Function marked as obsolete in AXL documentation

Command	Function Category
axl_ol_ol	R
axl_ol_ol2	R
axlAddAutoAssignNetAlgorithm	U
axlAddSelectAll	R
axlAddSelectBox	R
axlAddSelectName	R
axlAddSelectObject	R
axlAddSelectPoint	R
axlAddSimpleMoveDynamics	R
axlAddSimpleRbandDynamics	R
axlAddTaper	U

Command	Function Category
axlAirGap	R
axlAltSymbolList	R
axlAltSymbolOK	R
axlAltSymbolReplace	U
axlBackDrill	R, B
axlBackDrillGet	R
axlBondFingerDelete	U
axlBondWireDelete	U
axlBuildClassPopup	R
axlBuildSubclassPopup	R
axlCancelEnterFun	R
axlCancelOff	R
axlCancelOn	R
axlCancelTest	R
axlChangeLayer	R
axlChangeLine2Cline	S
axlChangeLineFont	S
axlChangeNet	S
axlChangeWidth	S
axlCheckString	R
axlClasses	R
axlClearDynamics	R
axlClearObjectCustomColor	I
axlClearSelSet	R
axlClipboardGetText	R
axlClipboardSetText	I
axlCmdList	R

Command	Function Category
axlCmdRegister	R
axlCmdUnregister	R
axlCnsAddVia	U
axlCNSAssemblyModeGet	R
axlCNSAssemblyModeSet	I
axlCnsAssignPurge	U, B
axlCnsClassTableChange	U
axlCnsClassTableCreate	U
axlCnsClassTableDelete	U
axlCnsClassTableFind	R
axlCnsClassTableSeek	R
axlCNSCreate	U
axlCNSCsetLock	U
axlCNSDelete	U
axlCnsDeleteClassClassObjects	U
axlCnsDeleteRegionClassClassObjects	U
axlCnsDeleteRegionClassObjects	U
axlCnsDeleteVia	U
axlCNSDesignModeGet	R
axlCNSDesignModeSet	U
axlCNSDesignValueCheck	R
axlCNSDesignValueGet	R
axlCNSDesignValueSet	U
axlCNSEcsetCreate	U
axlCNSEcsetDelete	U
axlCNSEcsetGet	R
axlCNSEcsetModeGet	R

Command	Function Category
axlCNSEcsetModeSet	U
axlCNSEcsetValueCheck	R
axlCNSEcsetValueGet	R
axlCNSEcsetValueSet	U
axlCNSGetAssembly	R
axlCNSGetDefaultMinLineWidth	R
axlCNSGetPhysical	R
axlCNSGetPinDelayEnabled	R
axlCNSGetPinDelayPVF	R
axlCNSGetSameNet	R
axlCNSGetSameNetXtalkEnabled	R
axlCNSGetSpacing	R
axlCnsGetViaList	R
axlCNSGetViaZEnabled	R
axlCNSGetViaZPVF	R
axlCNSIsCsetLocked	R
axlCNSIsLockedDomain	R
axlCnsList	R
axlCNSLockDomain	U
axlCNSMapClear	U
axlCNSMapUpdate	U
axlCnsNetFlattened	R
axlCNSPhysicalModeGet	R
axlCNSPhysicalModeSet	U
axlCnsPurgeAll	U
axlCnsPurgeCsets	U
axlCnsPurgeObjects	U

Command	Function Category
axlCNSSameNetModeGet	R
axlCNSSameNetModeSet	U
axlCNSSetAssembly	U
axlCNSSetPhysical	U
axlCNSSetPinDelayEnabled	U
axlCNSSetPinDelayPVF	I
axlCNSSetSameNet	U
axlCNSSetSameNetXtalkEnabled	I
axlCNSSetSpacing	U
axlCNSSetViaZEnabled	I
axlCNSSetViaZPVF	U
axlCNSSpacingMax	R
axlCNSSpacingMin	R
axlCNSSpacingModeGet	R
axlCNSSpacingModeSet	U
axlColorGet	R
axlColorLoad	I
axlColorOnGet	B
axlColorOnSet	B
axlColorPriorityGet	B
axlColorPrioritySet	B
axlColorSave	I
axlColorSet	I
axlColorShadowGet	R
axlColorShadowSet	I
axlCompAddPin	U
axlCompDeletePin	U

Command	**Function Category**
axlCompileSymbol	U
axlCompMovePin	U
axlComponentChangeClass	U
axlCompSetPinAttributes	U
axlConductorBottomLayer	R
axlConductorTopLayer	R
axlControlRaise	R
axlCopyObject	S
axlCopyProperties	U
axlcreate	U, B
axlCreateAttachment	U
axlCreateBondFinger	U
axlCreateBondWire	U
axlCreateDeviceFileTemplate	R
axlCreateWirebondGuide	S
axlCurrentDesign	R
axlCursorGet	R
axlCursorWarp	R
axlCustomColorObject	I
axlCVFColorChooserDlg	R
axlDB2Path	R
axlDBActiveShape	R
axlDBAddGroupObjects	U
axlDBAddProp	U
axlDBAltOrigin	R
axlDBAssignNet	S
axlDBChangeDesignExtents	U

Command	Function Category
axlDBChangeDesignOrigin	U
axlDBChangeDesignUnits	U
axlDBChangeText	S
axlDBCheck	U
axlDBCloak	S
axlDBControl	S
axlDBCopyPadstack	U
axlDBCreateCircle	S
axlDBCreateCloseShape	S
axlDBCreateComponent	U
axlDBCreateConceptComponent	U
axlDBCreateExternalDRC	U
axlDBCreateFilmRec	U, B
axlDBCreateGroup	U
axlDBCreateLine	S
axlDBCreateManyModuleInstances	U
axlDBCreateModuleDef	U
axlDBCreateModuleInstance	U
axlDBCreateNet	U
axlDBCreateOpenShape	S
axlDBCreatePadStack	U
axlDBCreatePath	S
axlDBCreatePin	U
axlDBCreatePropDictEntry	U
axlDBCreateRectangle	S
axlDBCreateShape	S
axlDBCreateSymbol	U

Command	Function Category
axlDBCreateSymbolAutosilk	U
axlDBCreateSymbolSkeleton	U
axlDBCreateSymDefSkeleton	U
axlDBCreateText	S
axlDBCreateVia	S
axlDBCreateVoid	S
axlDBCreateVoidCircle	S
axlDBDeleteProp	U
axlDBDeletePropAll	U
axlDBDeletePropDictEntry	U
axlDBDelLock	U
axlDBDisbandGroup	U
axlDBDisplayControl	I
axlDBDummyNet	R
axlDBDynamicShapes	U
axlDBFindByName	R
axlDBGetAttachedText	R
axlDBGetConnect	R
axlDBGetDesign	R
axlDBGetDesignUnits	R
axlDBGetDrillPlating	R
axlDBGetExtents	R
axlDBGetGroupFromItem	R
axlDBGetLayerType	R
axlDBGetLength	R
axlDBGetLock	R
axlDBGetLonelyBranches	R

Command	Function Category
axlDBGetManhattan	R
axlDBGetPad	R
axlDBGetPropDict	R
axlDBGetPropDictEntry	R
axlDBGetProperties	R
axlDBGetShapes	R
axlDBGetSymbolBodyExtent	R
axlDBGetTextBlockCount	R
axlDBGridGet	R
axlDBGridSet	I
axlDBGroupRename	U
axlDbidName	R
axlDBIgnoreFixed	S
axlDBIsBondingWireLayer	R, B
axlDBIsBondpad	R
axlDBIsBondwire	R
axlDBIsDiePad	R
axlDBIsDieStackLayer	R
axlDBIsFixed	R
axlDBIsPackagePin	R
axlDBIsPlatingbarPin	R
axlDBIsReadOnly	R
axlDBMemoryReclaim	R
axlDBOpenShape	S
axlDBPinPairLength	R
axlDBRefreshId	R
axlDBRemoveGroupObjects	U

Command	Function Category
axlDBSectorSize	U, B
axlDBSetLock	U
axlDBTextBlockCompact	U
axlDBTextBlockCreate	U
axlDBTextBlockFindName	R
axlDBTextBlockGetName	R
axlDBTextBlockSetName	U
axlDBTransactionCommit	S
axlDBTransactionMark	S
axlDBTransactionOops	S
axlDBTransactionRollback	S
axlDBTransactionStart	S
axlDBTuneSectorSize	U
axlDebug	R
axlDegToRad	R
axlDehighlightObject	I
axlDeleteAttachment	U
axlDeleteByLayer	U
axlDeleteFillet	U
axlDeleteObject	S
axlDeleteTaper	U
axlDesignFlip	R
axlDesignType	R
axlDetailLoad	U
axlDetailSave	S
axlDiffPair	U
axlDiffPairAuto	U

Command	Function Category
axlDiffPairDBID	R
axlDistance	R
axlDllCall	R
axlDllCallList	R
axlDllClose	R
axlDllDump	R
axlDllOpen	R
axlDllSym	R
axlDMBrowsePath	R
axlDMClose	R
axlDMDirectoryBrowse	R
axlDMFileBrowse	R
axlDMFileError	R
axlDMFileParts	R
axlDMFindFile	R
axlDMGetFile	R
axlDMLibraryFileNames	R
axlDMOpenFile	R
axlDMOpenLog	R
axldo	R
axlDrawObject	R
axlDRCGetCount	R
axlDRCItem	I
axlDRCUpdate	I
axlDRCWaive	U
axlDRCWaiveGetCount	R
axlDynamicsObject	R

Command	Function Category
axlEmail	R
axlEnterAngle	R
axlEnterBox	R
axlEnterEvent	R
axlEnterPath	R
axlEnterPoint	R
axlEnterPolar	R
axlEnterString	R
axlEraseObject	R
axlEventSetStartPopup	R
axlExportXmlDBRecords	U
axlExtentDB	R
axlExtentLayout	R, B
axlExtentSymbol	R, B
axlExtractMap	R
axlExtractToFile	R
axlfcreate	U, B
axlFillet	U
axlFilletConvert	U
axlFilmCreate	U
axlFindPath	R
axlFinishEnterFun	R
axlFlushDisplay	R
axlFormAutoResize	R
axlFormBNFDoc	R
axlFormBuildPopup	R
axlFormCallback	R

Command	Function Category
axlFormClearMouseActive	R
axlFormClose	R
axlFormColorize	R
axlFormCreate	R
axlFormDefaultButton	R
axlFormDisplay	R
axlFormFlexDoc	R
axlFormGetActiveField	R
axlFormGetField	R
axlFormGetFieldType	R
axlFormGridBatch	R
axlFormGridCancelPopup	R
axlFormGridDeleteRows	R
axlFormGridDoc	R
axlFormGridEvents	R
axlFormGridGetCell	R
axlFormGridInsertCol	R
axlFormGridInsertRows	R
axlFormGridNewCell	R
axlFormGridOption	R
axlFormGridOptions	R
axlFormGridReset	R
axlFormGridSelected	R
axlFormGridSelectedCnt	R
axlFormGridSetBatch	R
axlFormGridSetSelectRows	R
axlFormGridUpdate	R

Command	Function Category
axlFormIntroDoc	R
axlFormInvalidateField	R
axlFormIsFieldEditable	R
axlFormIsFieldVisible	R
axlFormListAddItem	R
axlFormListDeleteAll	R
axlFormListDeleteItem	R
axlFormListGetItem	R
axlFormListGetSelCount	R
axlFormListGetSelItems	R
axlFormListOptions	R
axlFormListSelAll	R
axlFormListSelect	R
axlFormMsg	R
axlFormRestoreField	R
axlFormSetActiveField	R
axlFormSetDecimal	R
axlFormSetEventAction	R
axlFormSetField	R
axlFormSetFieldEditable	R
axlFormSetFieldLimits	R
axlFormSetFieldVisible	R
axlFormSetInfo	R
axlFormSetMouseActive	R
axlFormTest	R
axlFormTitle	R
axlFormTreeViewAddItem	R

Command	Function Category
axlFormTreeViewChangeImages	R
axlFormTreeViewChangeLabel	R
axlFormTreeViewGetImages	R
axlFormTreeViewGetLabel	R
axlFormTreeViewGetParents	R
axlFormTreeViewGetSelectState	R
axlFormTreeViewLoadBitmaps	R
axlFormTreeViewSet	R
axlFormTreeViewSetSelectState	R
axlGeo2Str	R
axlGeoArcCenterAngle	R
axlGeoArcCenterRadius	R
axlGeoEqual	R
axlGeoPointInShape	R
axlGeoPointsEqual	R
axlGeoPointShapeInfo	R
axlGeoRotatePt	R
axlGetActiveLayer	R, B
axlGetActiveTextBlock	R, B
axlGetAlias	R
axlGetAllAttachmentNames	R
axlGetAllViaList	R
axlGetAllVisibleProfiles	R
axlGetAttachment	R
axlGetCmdSupplementalData	R
axlGetDieData	R
axlGetDieStackData	R

Command	Function Category
axlGetDieStackMemberSet	R
axlGetDieStackNames	R
axlGetDieType	R
axlGetDrawingName	R
axlGetDynamicsSegs	R
axlGetFindFilter	R
axlGetFuncKey	R
axlGetImpedance	R
axlGetIposerData	R
axlGetLastEnterPoint	R
axlGetLineLock	R
axlGetMetalUsageForLayer	R
axlGetModuleInstanceDefinition	R
axlGetModuleInstanceLocation	R
axlGetModuleInstanceLogicMethod	R
axlGetModuleInstanceNetExceptions	R
axlGetParam	R
axlGetSelSet	R
axlGetSelSetCount	R
axlGetSpacerData	R
axlGetTrapBox	R
axlGetVariable	R
axlGetVariableList	R
axlGetWireProfileColor	R
axlGetWireProfileDefinition	R
axlGetWireProfileDirection	R
axlGetWireProfileVisible	R

Command	**Function Category**
axlGetXSection	R, B
axlGRPDoc	R
axlGRPDrwBitmap	R
axlGRPDrwCircle	R
axlGRPDrwInit	R
axlGRPDrwLine	R
axlGRPDrwMapWindow	R
axlGRPDrwPoly	R
axlGRPDrwRectangle	R
axlGRPDrwText	R
axlGRPDrwUpdate	R
axlHighlightObject	R
axlHistory	R
axlHttp	R
axlIgnoreFixed	R
axlImpdedanceGetLayerBroadsideDPImp	R
axlImpdedanceGetLayerBroadsideDPWidth	R
axlImpdedanceGetLayerEdgeDPImp	R
axlImpdedanceGetLayerEdgeDPSpacing	R
axlImpdedanceGetLayerEdgeDPWidth	R
axlImpedance2Width	R
axlImportWireProfileDefinitions	U
axlImportXmlDBRecords	U
axlInTrigger	R
axlInTriggerFunc	R
axlIsAttachment	R
axlIsBetween	R

Command	Function Category
axlIsCustomColored	R
axlIsDBIDType	R
axlIsDebug	R
axlIsDummyNet	R
axlIsEtchLayer	R
axlIsFormType	R
axlIsGridCellType	R
axlIsHighlighted	R
axlIsitFill	R
axlIsLayer	R
axlIsLayerNegative	R
axlIsPinUnused	R
axlIsPointInsideBox	R
axlIsPointOnLine	R
axlIsPolyType	R
axlIsProductLineActive	R
axlIsProductStarted	R
axlIsProtectAlias	R
axlIsSymbolEditor	R
axlIsViewFileType	R
axlIsVisibleLayer	R
axlJournal	R
axlKillDesign	U
axlLastPick	R
axlLastPickIsSnapped	R
axlLayerCreateCrossSection	U, B
axlLayerCreateNonConductor	U

Command	**Function Category**
axlLayerDelete	U
axlLayerGet	R
axlLayerPriorityClearAll	I
axlLayerPriorityGet	I
axlLayerPriorityRestoreAll	I
axlLayerPrioritySaveAll	I
axlLayerPrioritySet	I
axlLayerSet	I
axlLayerViaLabel	R
axlLicDefaultVersion	R
axlLicFeatureExists	R
axlLicIsProductEnabled	R
axlLineSlope	R
axlLineXLine	R, B
axlLoadPadstack	U
axlLoadSymbol	U
axlLogHeader	R
axlMakeDynamicsPath	R
axlMapClassName	R
axlMatchGroupAdd	U
axlMatchGroupCreate	U
axlMatchGroupDelete	U
axlMatchGroupProp	U
axlMatchGroupRemove	U
axlMaterialGet	R
axlMathDotProduct	R
axlMemSize	R

Command	Function Category
axlMeterCreate	R
axlMeterDestroy	R
axlMeterIsCancelled	R
axlMeterUpdate	R
axlMidPointArc	R
axlMidPointLine	R
axlMiniStatusLoad	R
axlMiniStatusReset	R
axlMKS2UU	R
axlMKSAlias	R
axlMKSConvert	R
axlMKSStr2UU	R
axlMPythag	R
axlMsgCancelPrint	R
axlMsgCancelSeen	R
axlMsgClear	R
axlMsgContextClear	R
axlMsgContextFinish	R
axlMsgContextGet	R
axlMsgContextGetString	R
axlMsgContextInBuf	R
axlMsgContextPrint	R
axlMsgContextRemove	R
axlMsgContextStart	R
axlMsgContextTest	R
axlMsgPut	R
axlMsgSet	R

Command	Function Category
axlMsgTest	R
axlMUniVector	R
axlMXYAdd	R
axlMXYMult	R
axlMXYMultAdd	R
axlMXYSub	R
axlNetClassAdd	U
axlNetClassCreate	U
axlNetClassDelete	U
axlNetClassGet	R
axlNetClassRemove	U
axlNetEcsetValueGet	R
axlNetSched	U
axlNetsSched	U
axlOK2Void	R
axlOKToProceed	R
axlOpenDesign	U
axlOpenDesignForBatch	U
axlOpenFindFilter	R, B
axlOSBackSlash	R
axlOSControl	R
axlOSFileCopy	R
axlOSFileMove	R
axlOSNtp	R
axlOSSlash	R
axlPackageDesignCheckAddCategory	U
axlPackageDesignCheckAddCheck	U

Command	**Function Category**
axlPackageDesignCheckDrcError	U
axlPackageDesignCheckLogError	U
axlPadFigureTypes	R
axlPadOnLayer	R
axlPadstackEdit	U
axlPadstackSetType	U
axlPadstackToDisk	U
axlPadstackUsageTypes	R
axlPadSuppressGet	R
axlPadSuppressOkLayer	R
axlPadSuppressSet	U
axlPadUserMaskLayers	R
axlPathArcAngle	R
axlPathArcCenter	R
axlPathArcRadius	R
axlPathGetLastPathSeg	R
axlPathGetPathSegs	R
axlPathGetWidth	R
axlPathLine	R
axlPathOffset	R
axlPathSegGetArcCenter	R
axlPathSegGetArcClockwise	R
axlPathSegGetEndPoint	R
axlPathSegGetWidth	R
axlPathSetLineLock	R
axlPathStart	R
axlPathStartCircle	R

Command	Function Category
axlPdfView	R
axlPinExport	U
axlPinImport	U
axlPinPair	U
axlPinPairSeek	R
axlPinsOfNet	R
axlPolyErrorGet	R
axlPolyExpand	R
axlPolyFromDB	R
axlPolyFromHole	R
axlPolyMemUse	R
axlPolyOffset	R
axlPolyOperation	R
axlPPrint	R
axlPrintDbid	R
axlProtectAlias	R
axlPurgePadstacks	U
axlRadToDeg	R
axlRatsnestBlank	R
axlRatsnestDisplay	R
axlReadOnlyVariable	R
axlRecursiveDelete	R
axlRefreshSymbol	U
axlRegexpIs	R
axlRegionAdd	S
axlRegionCreate	U
axlRegionDelete	U

Command	Function Category
axlRegionRemove	S
axlRemoveNet	U
axlRenameDesign	U
axlRenameNet	U
axlRenameRefdes	U
axlReplacePadstack	U
axlReportList	R
axlReportRegister	R
axlReratNet	U
axlRunBatchDBProgram	U
axlSaveDesign	U
axlSaveEnable	U
axlScheduleNet	U
axlSegDelayAndZ0	R
axlSelect	R
axlSelectByName	R
axlSelectByProperty	R
axlSetActiveLayer	R, B
axlSetAlias	R
axlSetAllProfilesVisible	R
axlSetAttachment	U
axlSetBondWireProfile	U
axlSetDefaultDieInformation	U
axlSetDieData	U
axlSetDieStackData	U
axlSetDieType	U
axlSetDynamicsMirror	R

Command	Function Category
axlSetDynamicsRotation	R
axlSetFindFilter	R
axlSetFunckey	R
axlSetIposerData	U
axlSetLineLock	R
axlSetParam	I
axlSetPlaneType	U
axlSetRotateIncrement	R
axlSetSpacerData	U
axlSetSymbolType	U
axlSetVariable	R
axlSetVariableFile	R
axlSetWireProfileColor	I
axlSetWireProfileVisible	R
axlShapeAutoVoid	S
axlShapeChangeDynamicType	U
axlShapeDeleteVoids	S
axlShapeDynamicUpdate	S
axlShapeMerge	S
axlShapeRaisePriority	S
axlShell	R
axlShellPost	R
axlShoveItems	S
axlShoveSetParams	S
axlShowObject	R
axlShowObjectToFile	R
axlSingleSelectBox	R

Command	**Function Category**
axlSingleSelectName	R
axlSingleSelectObject	R
axlSingleSelectPoint	R
axlSleep	U
axlSmoothDesign	U
axlSmoothItems	S
axlSmoothSetParams	S
axlSnapToObject	R
axlSort	R
axlSpreadsheetClose	R
axlSpreadsheetDefineCell	R
axlSpreadsheetDoc	R
axlSpreadsheetGetCell	R
axlSpreadsheetGetRGBColorString	R
axlSpreadsheetGetRGBForNamedColor	R
axlSpreadsheetGetStyles	R
axlSpreadsheetGetWorksheets	R
axlSpreadsheetGetWorksheetSize	R
axlSpreadsheetInit	R
axlSpreadsheetRead	R
axlSpreadsheetReadDelimited	R
axlSpreadsheetSetCell	R
axlSpreadsheetSetCellProp	R
axlSpreadsheetSetColumnProp	R
axlSpreadsheetSetDocProp	R
axlSpreadsheetSetRowProp	R
axlSpreadsheetSetStyle	R

Command	Function Category
axlSpreadsheetSetStyleBorder	R
axlSpreadsheetSetStyleParent	R
axlSpreadsheetSetStyleProp	R
axlSpreadsheetSetWorksheet	R
axlSpreadsheetWrite	R
axlStrcmpAlpNum	R
axlStringCSVParse	R
axlStringRemoveSpaces	R
axlSubclasses	R
axlSubclassFormPopup	R
axlSubclassRoute	R
axlSubSelectAll	R
axlSubSelectBox	R
axlSubSelectName	R
axlSubSelectObject	R
axlSubSelectPoint	R
axlSymbolAttach	U
axlSymbolDetach	U
axlTechnologyType	R
axlTempDirectory	R
axlTempFile	R
axlTempFileRemove	R
axlTestPoint	S
axlText2Lines	R
axlTextOrientationCopy	R
axlTransformObject	S
axlTriggerClear	R

Command	Function Category
axlTriggerPrint	R
axlTriggerSet	R
axlUICmdPopupSet	R
axlUIColorDialog	R
axlUIConfirm	R
axlUIConfirmEx	R
axlUIControl	R
axlUIDataBrowse	R
axlUIEditFile	R
axlUIGetUserData	R
axlUIMenuChange	R
axlUIMenuDebug	R
axlUIMenuDelete	R
axlUIMenuDump	R
axlUIMenuFind	R
axlUIMenuInsert	R
axlUIMenuLoad	R
axlUIMenuRegister	R
axlUIMultipleChoice	R
axlUIPopupDefine	R
axlUIPopupSet	R
axlUIPrompt	R
axlUIViewFileCreate	R
axlUIViewFileReuse	R
axlUIViewFileScrollTo	R
axlUIWBeep	R
axlUIWBlock	R

Command	Function Category
axlUIWClose	R
axlUIWCloseAll	R
axlUIWDisableQuit	R
axlUIWExpose	R
axlUIWExposeByName	R
axlUIWHelpRegister	R
axlUIWIconify	R
axlUIWIsIconic	R
axlUIWIsWindow	R
axlUIWMove	R
axlUIWPerm	R
axlUIWPrint	R
axlUIWRedraw	R
axlUIWSetHelpTag	R
axlUIWSetParent	R
axlUIWShow	R
axlUIWSize	R
axlUIWTimerAdd	U
axlUIWTimerRemove	U
axlUIWUpdate	R
axlUIYesNo	R
axlUIYesNoCancel	R
axlUnfixAll	U
axlUnsetVariable	R
axlUnsetVariableFile	R
axlVersion	R
axlVersionIdGet	R

Command	Function Category
axlVersionIdPrint	R
axlViaZLength	R
axlVisibleDesign	R
axlVisibleGet	R
axlVisibleLayer	R
axlVisibleSet	R
axlVisibleUpdate	R
axlWFMAnyExported	R
axlWidth2Impedance	R
axlWindowBoxGet	R
axlWindowBoxSet	R
axlWindowFit	R
axlWriteDeviceFile	R
axlWritePackageFile	U
axlXSectionCopy	R
axlXSectionCreate	U
axlXSectionDelete	U
axlXSectionGet	R
axlXSectionLayerFunctions	R
axlXSectionLayerTypes	R
axlXSectionModify	U
axlXSectionSet	U
axlZoomBbox	R
axlZoomCenter	R
axlZoomControl	R
axlZoomFit	R
axlZoomInOut	R

Command	Function Category
axlZoomPoints	R
axlZoomToDbid	R
axlZoomWorld	R
bBoxAdd	R
copyDeep	R
isBoxp	R
lastelem	R
letStar	R
listnindex	R
movedown	R
moveup	R
parseFile	R
parseQuotedString	R
pprintln	R
propNames	R

Click here.