cadence®

Virtuoso Layout Suite SKILL Reference

Product Version IC23.1
November 2023

Printed in the United States of America.

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

Contents

8
Module Generator Functions 1611

9
Abstract Generator SKILL Functions 1921

16
Advanced Boolean Engine Functions

19
Virtuoso Automated Placement and Routing SKILL Functions

1

Basic Editing Functions

This topic provides a list of basic editing Cadence® SKILL functions associated with Virtuoso® Layout Suite.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

leAttachFig

leChopShape

leComputeAreaDensity

leComputeFigArea

leConvertInstToMosaic

leConvertMosaicToModgen

leConvertPolygonToPath

leConvertSelectedDynamicShapes

leConvertShapeToPathSeg

leConvertShapeToPolygon

leConvertTrimmedShapesToPRStyle

leCopyTemplatesToCellView

leCreateLayerField

leCreateNetField

leCycleSnapModes

leDecrementStopLevelByOne

leDefineExternalPins

leDefineInternalPins

leDefinePPPins

leDefineWeaklyConnectedPins

leHiAreaDensity

leHiAssignNet

leHiConvertMosaicToInst

leHiConvertMosaicToModgen

leHiEditSlot

leHiUnassignNet

leFlattenInst

leApplyAreaFunction

leApplyLastCopyTransform

leCLSCopyLayerShapes

leFullSelectFigOfSelSet

leGetAreaEstimationDisplayNames

leGetAreaEstimatorFunction

leGetAreaEstimatorParamList

leGetAreaEstimatorRunMode

leGetEditFigGroup

leGetEntryLayer

leGetObjectSelectable

leGetObjectVisible

leGetSnapOptions

leGetSnapTransform

leGetTechFormList

leGetValidLayerList

leGetValidPurposeList

leIsFigSelectable

leIsInstSelectable

leIsLayerSelectable

leIsLayerValid

leIsLayerVisible

leIsNewODCInfraEnabled

leIsPinSelectable

leIsPointInsideFig

leLSWGetBlockageSelectable

leLSWGetBlockageVisible

leLSWGetRoutingGridVisible

leLSWSetBlockageSelectable

leLSWSetBlockageVisible

leLSWSetRoutingGridVisible

leLSWSetTrackPatternVisible

lePrintHierarchyTree

leRaiseLSW

leRegAreaEstimator

leRegClusterBdyEstimator

leRegisterPPNetNameFn

leRegisterUseGravity

leRegUserLayerSelectionFilter

leRegUserObjectSelectionFilter

leRepeatCopyMoveStretch

leReportTrimmedShapesInCustomStyle

leResizeLSW

leSetAllGridObjectsVisible

leSetAllLayerSelectable

leSetAllLayerValid

leSetAllLayerVisible

leSetAllObjectsSelectable

leSetAllObjectsVisible

leSetAreaEstimatorParameters

leSetEditFigGroup

leSetEntryLayer

leSetFocusToEditableFieldsInStatusToolbar

leSetFormSnapMode

leHiCreateClusters

leHiCreateDonut

leHiCreateEllipse

leHiCreateGroup

leHiCreateGuardRing

leHiCreateInst

leHiCreateLabel

leHiCreateMPP

leHiCreatePath

leHiCreatePin

leHiCreatePinsFromLabels

leHiCreatePlacementArea

leHiCreatePolygon

leHiCreatePRBoundary

leHiCreateRect

leHiCreateRow

leHiCreateRuler

leHiCreateSnapBoundary

leHiCreateTaper

leHiCreateTrl

leHiCreateVia

leHiDelete

leHiDeleteAllAreaViewLevel

leHiDeleteAreaViewLevel

leHiDeleteShapeFromNet

leHiDescend

leHiDisplayPadOpeningInfoForm

leHiDisplayTechGraphForm

leHiEditDisplayOptions

leHiEditDRDOptions

leHiEditDRDRuleOptions

leHiEditDynamicMeasurementOptions

leHiEditEditorOptions

leHiEditInPlace

leHiEditProp

leHiFlatten

leHiFlip

leHiIncrementalViolation

leHiIncrementalViolationUpdater

leHiLayerGen

leHiLayerTap

leHiMakeCell

leHiMerge

leHiModifyCorner

leHiMousePopUp

leHiMove

leHiMoveOrigin

leHiOptionLayer

leHiPaste

leHiPlotQueueStatus

leHiPropagateNets

leHiQuery

leHiQuickAlign

leHiReShape

pteMapWindow

pteMinimizeSingleWindowPalette

pteMoveLayerSelection

pteMoveSingleWindowPalette

pteMPTSupportMode

pteOpenForm

ptePropagateSelectability

ptePropagateVisibility

pteRaiseSingleWindowPalette

pteRegisterUserLSManagerTrigger

pteRegisterUserSelectionTrigger

pteReloadLayerSet

pteResizeSingleWindowPalette

pteSaveAsLayerSet

pteSaveAsSynchronizedLayerSet

pteSaveConfig

pteSaveGDSNumber

pteSaveLayerSet

pteSaveLayerSetList

pteSaveLayerSetListInRepository

pteSaveLSWInfo

pteSaveToTechFile

pteSelect

pteSelectAll

pteSetActiveLpp

pteSetActiveLppColor

pteSetActiveLppColorLock

pteSetAllLayerSetMember

pteSetAllLSEnable

pteSetAllSelectable

pteSetAllStipple

pteSetAllValidity

pteSetAllVisible

pteSetBindkeys

pteSetConfig

pteSetFindModeOn

pteSetLayerSetColoredLpp

pteSetLayerSetMember

pteSetLppVisibleByString

pteSetLSActive

pteSetLSEnable

pteSetLSPosition

pteSetLSSelectable

pteSetLSVisible

pteSetMPTSupportMode

pteSetNoneLayerSetMember

pteSetNoneSelectable

pteSetNoneStipple

pteSetNoneValidity

pteSetNoneVisible

pteSetOnlySelectable

pteSetOnlySelectableWithDepth

pteSetOnlyVisible

pteSetOnlyVisibleWithDepth

pteSetOption

pteSetOptionString

pteSetRoutingDirection

pteSetSearchMatchCase

pteSetSearchMatchType

pteSetSearchOperator

pteSetSearchText

pteSetSelectable

pteSetSelectableWithDepth

pteSetShowLockedColors

pteSetShowUnlockedColors

pteSetStipple

pteSetValidity

pteSetVisible

pteSetVisibleWithDepth

pteSetWindowSynchro

pteShowAllTools

pteShowMPTLPP

pteShowRoutingLPP

pteShowUsedLPP

pteShowValidLPP

pteToggleAllLayerSetMember

pteToggleAllSelectable

pteToggleAllStipple

pteToggleAllTools

pteToggleAllValidity

pteToggleAllVisible

leSetLocalGridsVisible

leSetSnapPatternVisible

leSetWidthSpacingSnapPatternVisible

leToggleAllGravity

wspCheckActive

wspCreateWSP

wspCreateWSPByAttr

wspCreateWSPGroup

wspCreateWSSPDef

wspCreateWSSPDefByAttr

wspDeleteCellViewAllWSPData

wspDeleteMetalFill

wspDumpToFile

wspGetLineEndGrids

wspGetWSSPDefLP

wspRegionFindByLayer

wspRegionFindByWSSPDef

wspRegionGetActivePattern

wspRegionGetAllowedPatternGroups

wspRegionGetAllowedPatterns

wspRegionGetWSSPDef

wspSetWSSPDefRegionPurpose

wspSPDefFind

wspWSPFindByName

wspWSPGetFlatAttr

wspWSPGroupFindByName

wspWSSPDefAddToAllowedPatternGroups

Basic Layout Editing SKILL Function and Menu Command Mappings

The following table summarizes the procedural and interactive SKILL functions associated with the basic layout editing menu commands.

Interactive SKILL Function	Procedural SKILL Function	Menu Command
hiLayerDispMainForm	—	*CIW – Tools – Technology File Manager – Edit Layers*
leCloseWindow	hiCloseWindow	*File – Close*
leDesignSummary	—	*File – Summary*
leEditDesignProperties	—	*File – Properties*
leHiAbout	—	*Help – About Layout*
leHiAddShapeToNet	—	*Connectivity – Nets – Add Shape*
leHiAlign	—	*Edit – Advanced – Align*
leHiAttach	leAttachFig	*Edit – Advanced – Attach/Detach*
leHiCellviewTrackPatterns	—	*Create – P&R Objects – Track Patterns*
leHiChop	leChopShape	*Edit – Basic – Chop*
leHiClearMeasurement	leClearAllMeasurement	*Tools – Clear All Measurements*
leHiConvertShapeToPolygon	leConvertShapeToPolygon	*Edit – Advanced – Convert to Polygon*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiCopy	dbCopyFig	*Edit – Basic – Copy*
leHiCreateBend	geCreateBend	*Create – Microwave – Bend*
leHiCreateCircle	dbCreateEllipse	*Create – Shape – Circle*
leHiCreateDonut	dbCreateDonut	*Create – Shape – Donut*
leHiCreateEllipse	dbCreateEllipse	*Create – Shape – Ellipse*
leHiCreateGuardRing	—	*Create – Guard Ring*
leHiCreateInst	dbCreateSimpleMosaic dbCreateInst	*Create – Instance*
leHiCreateLabel	dbCreateLabel	*Create – Label*
leHiCreatePath	leCreatePath	*Create – Shape – Path*
leHiCreatePin	leCreatePin	*Create – Pin*
leHiCreatePinsFromLabels	—	*Tools – Create Pins From Labels*
leHiCreatePolygon	dbCreatePolygon	*Create – Shape – Polygon*
leHiCreateRect	dbCreateRect	*Create – Shape – Rectangle*
leHiCreateMeasurement	leCreateMeasurement	*Tools – Create Measurement*
leHiCreateTaper	geCreateTaper	*Create – Microwave – Taper*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiCreateTrl	geCreateTrl	*Create – Microwave – Trl*
leHiCreateVia		
leHiDelete	dbDeleteObject	*Edit – Basic – Delete*
leHiDeleteAllAreaViewLevel	geDeleteAllAreaViewLevel	*View – Area Display – Delete All*
leHiDeleteAreaViewLevel	geDeleteAreaViewLevel	*View – Area Display – Delete*
leHiDeleteShapeFromNet	—	*Connectivity – Nets – Remove Shape*
leHiDescend	leDescend	*Edit – Hierarchy – Descend*
leHiEditDisplayOptions	envGetVal envSetVal	*Options – Display*
leHiEditDynamicMeasurementOptions	—	*Options – Dynamic Measurements*
leHiEditDRDOptions	—	*Options – DRD Edit*
leHiEditEditorOptions	envGetVal envSetVal	*Options – Editor*
leHiEditInPlace	leEditInPlace	*Edit – Hierarchy – Edit In Place*
leHiEditProp	—	*Edit – Basic – Properties*
leHiFlatten	leFlattenInst	*Edit – Hierarchy – Flatten*
leHiFlip	—	*Edit – Flip*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiLayerGen	leLayerAnd leLayerAndNot leLayerOr leLayerSize leLayerXor	*Tools – Layer Generation*
leHiLayerTap	—	*LSW: Edit – Tap*
leHiMakeCell	leMakeCell	*Edit – Hierarchy – Make Cell*
leHiMarkNet	—	*Connectivity – Net – Mark*
leHiMerge	leMergeShapes	*Edit – Basic – Merge*
leHiModifyCorner	leModifyCorner	*Edit – Advanced – Modify Corner*
leHiMove	dbMoveFig	*Edit – Basic – Move*
leHiMoveOrigin	leMoveCellViewOrigin	*Edit – Advanced – Move Origin*
leHiOptionLayer	—	*LSW: Edit – Option*
leHiPaste	lePasteFigs	*Edit – Basic – Paste*
leHiPlotQueueStatus	—	*File – Print Status*
leHiPropagateNets	—	*Connectivity – Nets – Propagate*
leHiReShape	—	*Edit – Advanced – Reshape*
leHiRemasterInstances	leRemasterInstances	*Tools – Remaster Instances*

Interactive SKILL Function	Procedural SKILL Function	Menu Command
leHiRepeatCopy	—	*Edit – Repeat Copy*
leHiRotate	—	*Edit – Basic – Rotate*
leHiSearch	leSearchHierarchy leReplace	*Tools – Find/ Replace*
leHiSetAreaViewLevel	geSetAreaViewLevel	*View – Area Display – Set*
leHiSetRefPoint	leSetRefPoint	
leHiShowSelSet	—	*View – Show Selected Set*
leHiSize	leSizeShape	*Edit – Advanced – Size*
leHiSplit	leSplitShape	*Edit – Advanced – Split*
leHiStretch	leStretchShape	*Edit – Basic – Stretch*
leHiSubmitPlot	lePlot	*File – Print*
leHiSummary	—	*File – Summary*
leHiTree	—	*Edit – Hierarchy – Tree*
leHiUnmarkNet	—	*Connectivity – Nets – Unmark*
leHiYank	leYankFigs	*Edit – Basic – Yank*
leiDiscardEdits	—	*File – Discard Edits*

Environment Variable Functions

You can use SKILL functions to set environment variable values and get the current environment variable settings. For example, you can use `leEnvLoad` to load Virtuoso Layout Suite environment variables from the `.cdsenv` files.

Related Topics

leEnvLoad

leGetEnv

leSetEnv

leEnvLoad

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

Description

Loads the Virtuoso Layout Suite environment variables from the `.cdsenv` files. This file must be in your home directory. The variables are read into memory, but the windows are not updated with the new values. When new windows are created, they assume the new values.

Value Returned

`t`	The environment variables are loaded.
`nil`	The environment variables are not loaded.

Related Topics

Environment Variable Functions

Layout XL Basic Editing Environment Variables

leGetEnv

```
leGetEnv(
     t_name
     )
     => g_value / nil
```

Description

Returns the value currently assigned to the layout environment properties with the variable *t_name*.

Arguments

t_name	Name of the layout environment variable.

Value Returned

g_value	The variable *t_name*.
nil	The specified variable does not exist.

Example

Returns the value of the variable gravityOn.

```
leGetEnv( "gravityOn" )
```

Related Topics

Environment Variable Functions

leSetEnv

```
leSetEnv(
     t_name
     g_value
     )
     => t / nil
```

Description

Sets the layout environment properties with the variable *t_name* to the value *g_value*.

Arguments

t_name	Name of the layout environment property variable assigned the value *g_value*. Valid Values: any valid SKILL expression
g_value	Value of the variable *t_name*. Valid Values: any value that matches the value type defined for the variable

Value Returned

t	The value is assigned.
nil	The value is not assigned.

Example

Turns gravity off in the current window.

```
leSetEnv( "gravityOn" nil )
```

Related Topics

Environment Variable Functions

Layout Viewer Functions

The Layout Viewer functions enable you to check the availability of the Layout Viewer application.

- leIsLayoutViewerAppEnabled: Checks whether the layout viewer application is available.
- leIsLayoutViewerWindow: Checks whether the given window has the Layout Viewer application installed on it.

leIsLayoutViewerAppEnabled

```
leIsLayoutViewerAppEnabled(
     w_windowId
     )
     => t / nil
```

Description

Checks whether the layout viewer application is available.

Arguments

None

Value Returned

`t`	The layout viewer application is available.
`nil`	The layout viewer application is not available.

Example

Returns `t` if the layout viewer application is available.

```
leIsLayoutViewerAppEnabled()
```

leIsLayoutViewerWindow

```
leIsLayoutViewerWindow(
     w_windowId
     )
     => t / nil
```

Description

Checks whether the given window has layout viewer application installed on it.

Arguments

w_windowId	Window ID of the window.

Value Returned

t	The *w_windowId* has layout viewer application installed.
nil	The *w_windowId* does not have the layout viewer application installed.

Example

Returns `t` if the window `winId` has layout viewer application installed on it.

```
leIsLayoutViewerWindow(winId)
```

Object Creation and Plotting Functions

You can create a variety of objects and templates for multipart paths with Virtuoso Layout Suite. The following table describes the objects that you can create.

Object	Description
Arrays	Arrays are created using `dbMosaic` objects. You can create simple mosaics only. Although you can manipulate the location and orientation of complex mosaics, you cannot create them, nor can you change their composition. You must use the structure compiler tool for these operations.
Circles	Circles are created using `dbEllipse` objects. Circles are stored as ellipses, where the bounding box of the ellipse is a square. A circle does not have to remain a circle; you can stretch it into an ellipse.
Contacts	Contacts are created using `dbInst` objects. Contacts are defined in the technology file and must be parameterized cells with specific parameter names.
Donuts	Donuts are created using `dbDonut` objects.
Ellipses	Ellipses are created using `dbEllipse` objects.
Instances	Instances are created using `dbInst` objects.
Labels	Labels are created using `dbLabel` objects.
Multipart Path Templates	One way to define a template for a multipart path is by specifying the `leDefineMPPTemplate` function in an ASCII file. A multipart path is a single relative object design (ROD) object consisting of one or more parts at level zero in the hierarchy on the same or on different layers.
Paths	Paths are created using `dbPath` objects. You can use the *Path* command to create multi-layer paths, which are multiple paths on different layers with contact instances connecting the different layers.

Object	Description
Pins	Rectangle, polygon, dot, and symbolic (instance) type pins can be created. They are created as `dbRect`, `dbPolygon`, `dbDot`, or `dbInst` types, with `dbPins` attached to them. From a database point of view, a pin is a logical object, not a physical object; the object seen is just the figure used to represent the pin. Pins can be created on any layer and can be of any size. The system prompts you for the terminal name.
Polygons	Polygons are created using `dbPolygon` objects.
Procedural Access	All figure types can be created procedurally. Many of these procedures are identical to database functions and maintain a consistent naming convention.
Rectangles	Rectangles are created using `dbRect` objects.
Tapered Transmission Lines	Tapered transmission lines are a special type of transmission line: they are a single transmission line segment that has different widths at either end. The transition from one width to the other can be either linear or exponential. Tapered transmission lines are represented in the database in the same way as transmission lines.
Transmission Line Bends	Transmission line bends are special types of transmission lines. They contain three points and, therefore, define two segments and one bend. Unlike normal transmission lines, the two segments of a transmission line bend can have different widths. Transmission line bends are represented in the database in the same way as transmission lines.
Transmission Lines	Transmission lines are similar to paths, except that you can specify how bends in transmission lines are built. You can specify standard bends, chamfered bends, or radial bends. Transmission lines are useful in extremely high-frequency circuits.

Related Topics

leCreateAutoPin

leCreatePath

leCreatePin

leHiCreateSlot

leDefineMPPTemplate

lePlot

leCreateAutoPin

```
leCreateAutoPin(
     d_cellViewId
     l_point
     t_termName
     t_termDir
     [ l_lpp ]
     [ g_pinSnapToWireEnd ]
     [ x_depth ]
     [ g_truncateWire ]
     )
     => d_pinShapeId / nil
```

Description

Creates a pin shape in cellview *d_cellViewId*.

Arguments

d_cellViewId	The database ID of the cellview in which the pin is created.
l_point	The point at which to create the rectangle pin.
t_termName	Name of the terminal to which the pin is attached. If no such terminal exists, one is created. If the terminal must be created, it is created on a net with the same name as the terminal. Valid Values: any valid SKILL expression.
t_termDir	Specifies the I/O type (direction) of the pin. Valid Values: input, output, inputOutput, switch, or jumper.
l_lpp	Specifies the list of layer purpose pairs. If specified, the function creates the pin on the shapes on the specified LPP.
g_pinSnapToWireEnd	Creates a pin on the end of a path segment. The valid values are `t` and `nil`. The default is `t`.
	When set to `nil`, the pin is created in the center of the path segment.
x_depth	Creates the pin at the specified depth. This argument is considered only when the `enablePinEnhancements` environment variable is set to `t`.
g_truncateWire	Truncates the path segment to the PR boundary if the path segment extends beyond the PR boundary.

Value Returned

d_pinShapeId	The object ID for the pin if the pin is created.
nil	The pin is not created.

Examples

In the following example, a pin is created in cellview `cell1` at `10:10`. The pin is attached to terminal `term1`, and the access direction of the pin is `input`. Returns the object ID of the pin.

```
leCreateAutoPin( cell1 list(10:10) "term1" "input" )
```

In the following example, a pin is created in cellview `cell1` at `10:10`. The pin is attached to terminal `term1`, and the access direction of the pin is `input`. The pin is created on shapes on layer-purpose `poly1 drawing`, and is present at `level 0`. If the underlying shape is a path segment, the pin is created at the center and is abutted to the PR boundary if more than half of the path segment extends beyond the PR boundary. Returns the object ID of the pin.

```
leCreateAutoPin(cell1 list(10:10) "term1" "input" list("poly1" "drawing") nil 0 t)
```

leCreatePath

```
leCreatePath(
     d_cellViewId
     l_layerPurposePair
     l_points
     n_width
     [ t_pathStyle ]
     [ n_offset ]
     [ t_justification ]
     )
     => d_pathId / nil
```

Description

Creates a path in cellview *d_cellViewId* on the specified layer with point list *l_points* and width *n_width*.

Arguments

d_cellViewId	The database ID of the cellview in which the path is created.
l_layerPurposePair	Layer-purpose pair of the path. Valid Values: any valid layer name and purpose if other than `drawing`
l_points	List of coordinates used to create the path.
n_width	Specifies the width of the current segment of the path. Valid Values: any non-negative number in user units
t_pathStyle	Specifies how the path segment ends are built for the current path segment. Valid Values: `truncateExtend`, `extendExtend`, `roundRound`, or `varExtendExtend`
n_offset	Specifies the offset from the points you enter to the path. Valid Values: a positive or negative number
t_justification	Specifies whether the path points correspond to the centerline of the path, its left edge, or its right edge. The offset is relative to the specified edge. Valid Values: `left`, `center`, or `right`

Value Returned

d_pathId	The object ID of the path if the path is created.
`nil`	The path is not created.

Example

Creates a path in cellview `cell2` on layer `metal1R` with the coordinates `0:0`, `10:0`, `10:10` and returns the object ID of the path. The path has a width of `5`.

```
leCreatePath( cell2 list("metal1R") list(0:0 10:0 10:10) 5 )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to enter the layer, coordinates, width, and options. If you do not specify *w_windowId*, the current window is used.

```
leHiCreatePath( [ w_windowId ] ) => t / nil
```

leCreatePin

```
leCreatePin(
     d_cellViewId
     l_layerPurposePair
     t_shape
     l_points
     t_termName
     t_termDir
     l_accessDir
     )
     => d_pinShapeId / nil
```

Description

Creates a pin shape in cellview *d_cellViewId* on the specified layer.

Arguments

d_cellViewId	The database ID of the cellview in which the pin is created.
l_layerPurposePair	Specifies the layer-purpose pair on which to create the pin. Valid Values: any valid layer and purpose name
t_shape	Type of shape to use for the pin. Valid Values: `rectangle`, `polygon`, or `circle`.
l_points	List of points to use for the creation of the pin. Valid Values: two for a rectangle and more than two for a polygon pin.
t_termName	Name of the terminal to which the pin is attached. If no such terminal exists, one is created. If the terminal must be created, it is created on a net with the same name as the terminal. Valid Values: any valid SKILL expression
t_termDir	Specifies the I/O type (direction) of the pin. Valid Values: `input`, `output`, `inputOutput`, `switch`, or `jumper`
l_accessDir	List of access directions of the pin. This field is used only for rectangular pins. Valid Values: `none`, `top`, `bottom`, `left`, or `right`

Value Returned

d_pinShapeId	The object ID for the pin if the pin is created.
`nil`	The pin is not created.

Example

Creates a pin in the current cellview on layer `metal1R` from `0:0` to `10:10`. The pin is attached to terminal `term1`; the I/O type of the pin `input` and the access directions of the pin are `left` and `right`. Returns the object ID of the pin.

```
leCreatePin( geGetEditRep() "metal1R" "rectangle" list(0:0 10:10) "term1" "input"
list("left" "right") )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to enter the terminal name, pin shape, I/O type, and access direction for the pin. If you do not specify *w_windowId*, the current window is used.

```
leHiCreatePin( [ w_windowId ] ) => t / nil
```

leDefineMPPTemplate

```
leDefineMPPTemplate(
          ?techId              d_techId
          ?name                S_name
          ?layer               list(layerName layerPurpose)
          [ ?width             n_width ]
          [ ?justification     S_justification ]
          [ ?offset            n_offset ]
          [ ?endType           S_endType ]
          [ ?beginExt          n_beginExt ]
          [ ?endExt            n_endExt ]
          [ ?choppable         g_choppable ]

          [ ROD Connectivity Arguments ]
          [ ?offsetSubPath     l_offsetSubpathArgs... ]
          [ ?encSubPath        l_encSubpathArgs... ]
          [ ?subRect           l_subrectArgs... ]

     ) ; end leDefineMPPTemplate
     => t / nil

          ; ROD Connectivity Arguments
          [ ?termIOType              S_termIOType ]
          [ ?pin                     g_pin ]
          [ ?pinAccessDir            tl_pinAccessDir ]
          [ ?pinLabel                g_pinLabel ]
          [ ?pinLabelHeight          n_pinLabelHeight ]
          [ ?pinLabelLayer           txl_pinLabelLayer ]
          [ ?pinLabelFont            S_pinLabelFont ]
          [ ?pinLabelDrafting        g_pinLabelDrafting ]
          [ ?pinLabelOrient          S_pinLabelOrient ]
          [ ?pinLabelOffsetPoint     l_pinLabelOffsetPoint ]
          [ ?pinLabelJust            S_pinLabelJust ]
          [ ?pinLabelRefHandle       S_pinLabelRefHandle ]
          ; end of ROD Connectivity Arguments

          ;l_offsetSubpathArgs
          list(
               list(
                    ?layer             list(layerName layerPurpose)
                    [ ?width           n_width ]
                    [ ?sep             n_sep ]
                    [ ?justification   S_justification ]
                    [ ?beginOffset     n_beginOffset ]
                    [ ?endOffset       n_endOffset ]
                    [ ?choppable       g_choppable ]
                    [ Repeat ROD Connectivity Arguments here ]
                    ) ;End of first offset subpath list
                    ...
               ) ;End of offset subpath lists
```

```
;End of l_offsetSubpathArgs

;l_encSubpathArgs
list(
     list(
          ?layer             list(layerName layerPurpose)
          [ ?enclosure       n_enclosure ]
          [ ?beginOffset     n_beginOffset ]
          [ ?endOffset       n_endOffset ]
          [ ?choppable       g_choppable ]
          [ Repeat ROD Connectivity Arguments here ]
          ) ;End of first enclosure subpath list
          ...
     ) ;End of enclosure subpath lists
;End of l_encSubpathArgs

;l_subrectArgs
list(
     list(
          ?layer              list(layerName layerPurpose)
          [ ?width            n_width ]
          [ ?length           n_length ]
          [ ?gap              S_gap ]
          [ ?sep              n_sep ]
          [ ?justification    S_justification ]
          [ ?beginOffset      n_beginOffset ]
          [ ?endOffset        n_endOffset ]
          [ ?beginSegOffset   n_beginSegmentOffset
          [ ?endSegOffset     n_endSegmentOffset
          [ ?space            n_space ]
          [ ?choppable        g_choppable ]
          [ Repeat ROD Connectivity Arguments here ]
          [?diagonal g_diagonalSubRect]
          ) ;End of first subrectangle list
          ...
     ) ;End of subrectangle lists
;End of l_subrectArgs
```

Description

Provides the structure for defining the field names and default values for one multipart path (MPP) in an ASCII file. This information is referred to as an MPP *template*. A multipart path is a single relative object design (ROD) object consisting of one or more parts at level zero in the hierarchy on the same or on different layers. The purpose of an MPP template is to create MPPs in layout cellviews using predefined values. You can define any number of MPP templates in a single ASCII file by specifying `leDefineMPPTemplate` once for each MPP template; each template must be identified by a template name (*`t_name`*) that is unique within the ASCII file.

Arguments

d_techId	The database ID of the technology library associated with the current cellview.
S_name	Character string enclosed in double quotation marks specifying the name of the MPP template. The name must be unique within the MPP templates in the ASCII file. Do not assign the name `New`; it is a reserved name. Default: `none`

All other arguments for `leDefineMPPTemplate` have the same names, definitions, valid values, and default values as the arguments for the `rodCreatePath` function. For a detailed description of each argument, see the Arguments section of the rodCreatePath function.

Value Returned

`t`	Returned if the argument list is created successfully.
`nil`	Returned if the argument list is not created successfully.

ASCII MPP Template File

When you load an ASCII MPP template file, the system checks the names of the ASCII MPP templates against the names of MPP templates that already exist in the technology file in virtual memory. If there is a name conflict, the system displays a message in the CIW saying that an MPP template in the ASCII file is replacing an existing MPP template. The existing template is overwritten in virtual memory only. If you do not want to overwrite the original MPP template in your binary technology library on disk, do not save changes to the technology file. You can also avoid overwriting existing templates by changing the matching names in your ASCII file. For more information, see Template templateName is replacing an existing template by same name.

The system merges the ASCII MPP template definitions into the technology file in virtual memory. When you choose the Virtuoso Layout Suite *Create – Multipart Path* command (and press F3), the system displays the names of all MPP templates *MPP Template* list in the Create Multipart Path form.

You can load ASCII files in any of the following ways:

- In Virtuoso Layout Suite, with the *Create – Multipart Path* command, using the *Load Template* option.
- In the CIW, with the following statement:

```
load "pathToFile/templateFilename.il"
```

For example,

```
load "../skill/mppTemplates.il"
```

- In the `.cdsinit` file

  You can identify ASCII MPP template files in your `.cdsinit` file. The system automatically executes the `.cdsinit` file when the Cadence® design framework II product starts, and loads the identified files.

- In the `libInit.il` file

  You can attach ASCII MPP template files to your library by identifying them in a `libInit.il` file. The system automatically executes the `libInit.il` file when a library is opened and loads attached files.

  Note: Avoid using graphical SKILL functions such as `hiSetBindKey` in your `libInit.il` file; doing so will result in an error. For more information, see What to Avoid in the libInit.il File.

  To attach a file to a library:

  - Place the file within the library directory.
  - In the library directory, create or update a file named `libInit.il`.
  - Within the `libInit.il` file, write a `load` statement as follows:

    ```
    load "pathToFile/templateFilename.il"
    ```

Loading MPP templates from an ASCII file affects only the temporary version of your technology library in virtual memory. If you want the loaded templates and changes you make to their values with the Create Multipart Path form to persist beyond the end of the current editing session, you must save the changes to your binary technology library on disk before you exit the software. Remember that MPP templates loaded from ASCII files overwrite templates with matching names in the technology file in virtual memory, and when you save changes to your technology file, the virtual memory version overwrites your technology library on disk.

You can create an ASCII MPP template file:

- With a text editor
- With Virtuoso Layout Suite by using the *Save Template* option on the Create Multipart Path form and saving to ASCII

The following example shows an ASCII file containing the definitions for two MPP templates.

- When you create an ASCII template file with a text editor, you need to substitute your own library and technology file names for the variables *myLibraryName* and *myTechFileName* in the example.
- If you enter an invalid technology file name, the system displays a warning message in the CIW and no templates are loaded.
- When you save the values in the Create Multipart Path form to ASCII, the system replaces the *myLibraryName* and *myTechFileName* variables with the name of the current library and the name of the current technology file.

If you load an ASCII file that was created with the Create Multipart Path form, make sure you are using the same library and technology file that was current when the ASCII file was created.

```
;; This procedure identifies the technology library and technology file names, and
;; opens the technology file in read-only mode.If the technology library does not
;; open, the procedure displays an error message and exits.If the technology library
;; opens,the procedure continues, and reads the leDefineMPPTemplate functions,
;; each of which define the fields and values for a multipart path (MPP).


prog(
    ( tech techLibName techFileName)
        techLibName  =         "myLibraryName"
        techFileName =         "myTechFileName"
        tech         =         techOpenTechFile( techLibName techFileName "r" )

        if(tech==nil
        then
          return(nil)
        ) ; end if

    ; Template1 defines an MPP consisting of only a master path.

    leDefineMPPTemplate(
        ?techId            tech
        ?name             "template1"
        ?layer             list("poly1" "drawing")
        ?width             0.600000
        ?choppable         t
        ?endType           "flush"
        ?justification     "center"
        ?offset            0.700000
        ?termIOType        "switch"
        ?pin               t
        ?pinAccessDir      list( "bottom" "left" "right" )

    ) ; end leDefineMPPTemplate function for template1

    ; Template2 defines an MPP consisting of a master path, three offset subpaths,
    ; three enclosure subpaths, and three sets of subrectangles.

    leDefineMPPTemplate(
               ; Master path
               ?techId          tech
               ?name            "Template2"
```

```
      ?layer           "cellBoundary"
      ?width           0.600000

; Three offset subpaths

?offsetSubPath
      list(
                  list(
                        ?layer           list("pwell" "drawing")
                        ?width           1.000000
                        ?choppable       t
                        ?sep             2.000000
                  ) ; end of first sublist
                  list(
                        ?layer           list("poly1" "drawing")
                        ?width           1.000000
                        ?choppable       t
                        ?sep             4.000000
                  ) ; end of second sublist
                  list(
                        ?layer           list("metal1" "drawing")
                        ?width           1.000000
                        ?choppable       t
                        ?sep             6.000000
                  ) ; end of third sublist
      ) ; end of offset subpath lists

; Three enclosure subpaths

?encSubPath
      list(
                  list(
                        ?layer           list("metal2" "drawing")
                        ?enclosure       0.100000
                        ?choppable       t
                        ?beginOffset     0.100000
                        ?endOffset       0.200000
      ) ; end of first sublist
                  list(
                        ?layer           list("metal3" "drawing")
                        ?enclosure       0.150000
                        ?choppable       t
                        ?beginOffset     0.200000
                        ?endOffset       0.300000
      ) ; end of second sublist
                  list(
                        ?layer           list("ndiff" "drawing")
                        ?enclosure       0.180000
                        ?choppable       t
                        ?beginOffset     0.300000
                        ?endOffset       0.400000
                  ) ; end of third sublist
      ) ; end of enclosure subpath lists

; Three sets of subrectangles

?subRect
      list(
                  list(
                        ?layer           list("pdiff" "drawing")
                        ?width           0.500000
```

```
                                    ?length          0.600000
                                    ?choppable       t
                                    ?sep             0.200000
                                    ?space           1.000000
                                ) ; end of first sublist
                                list(
                                    ?layer           list("pwell" "drawing")
                                    ?width           0.600000
                                    ?length          0.800000
                                    ?choppable       t
                                    ?sep             0.500000
                                    ?space           3.000000
                                ) ; end of second sublist
                                list(
                                    ?layer           list("poly1" "drawing")
                                    ?width           0.800000
                                    ?length          1.000000
                                    ?choppable       t
                                    ?sep             2.000000
                                    ?space           5.000000
                                ) ; end of third list
                        ) ; end of subrectangle lists
    ) ; end leDefineMPPTemplate function for template2

) ; end prog
```

Differences Between leDefineMPPTemplate and rodCreatePath Syntax

The syntax for specifying an MPP template with `leDefineMPPTemplate` is based on the syntax of the `rodCreatePath` function. Most arguments are the same and have the same default values. The following are the differences between the syntax for an MPP template specified with `leDefineMPPTemplate` and `rodCreatePath`:

- For some arguments, the data type is more restricted for `leDefineMPPTemplate` than it is for `rodCreatePath`:
  - For the `?layer` argument, `leDefineMPPTemplate` requires a list; `rodCreatePath` accepts a text string or an integer, or a list specifying the layer or layer-purpose pair.
  - For arguments with the data type *`S_`*, `leDefineMPPTemplate` requires a character string (the data type is *`t_`* for text); `rodCreatePath` allows either a character string or a symbol.
- For the remaining arguments that `leDefineMPPTemplate` and `rodCreatePath` have in common, the argument definitions, valid values, and default values are identical.
- `leDefineMPPTemplate` has one additional argument: *`t_name`*
- You cannot specify an expression as a value for any of the `leDefineMPPTemplate` arguments.

- `leDefineMPPTemplate` does not contain the following `rodCreatePath` arguments:

d_cvId	The current cellview is used.
S_name	The system generates a default ROD name when you create an MPP in a cellview; you can change it in the *ROD Name* field on the Create ROD Multipart Path form or in the Edit Properties form.
S_netName	When the MPP template you choose includes the definition of pin connectivity, to create an MPP in a layout cellview, you must enter a value for the *Net Name* field in the Create ROD Multipart Path form.
S_termName	When the MPP template includes the definition of pin connectivity, the system assigns the net name you enter for terminal name.
l_pts	Click in the layout cellview to specify the point list.
l_prop	There is no property field in the Create Multipart Path forms, so this argument has been omitted.

- `leDefineMPPTemplate` does not contain the `rodCreatePath` arguments listed below because these arguments represent an alternate way of specifying a point list for an MPP, and the point list varies for each occurrence of an MPP within a layout cellview.

  dl_fromObj *txf_size*
  l_startHandle *l_endHandle*

Related Topics

rodCreatePath.

ROD Connectivity Arguments

Offset Subpath Arguments

Enclosure Subpath Arguments

Subrectangle Arguments

leHiCreateSlot

```
leHiCreateSlot(
     [ w_windowId ]
     )
     => t / nil
```

Description

Runs the *Create Slot* command in the specified window. You are prompted to select a rectangle or path to which you want to add slots. Press `F3` to open the *Create Slot* form and specify the settings for creating slots. If *`w_windowId`* is not specified, the current window is used.

Arguments

`w_windowID`	Window ID of the window.

Value Returned

`t`	The slot was created.
`nil`	The slot was not created.

lePlot

```
lePlot(
     [ t_fileName ]
     )
     => t / nil
```

Description

Generates the plot defined in the *t_fileName* plot template file. Plot options are stored in a disembodied property list called `lePlotOptions`. When `lePlot()` loads the plot template file, it reads the plot options from the `lePlotOptions` property list. There are two ways to create the plot template file: use `vi`, or save the options in the Submit Plot form to a file. The plot template file is used by `lePlot()` and by the *Load* command in the Submit Plot form. However, the following options are ignored by the *Load* command: `library`, `cell`, `view`, and `plotSize`.

Arguments

t_fileName	A plot template file that stores plot options in a disembodied property list named `lePlotOptions`.

Value Returned

`t`	The operation was successful.
`nil`	The operation was not successful.

Examples

Plots a cellview as specified in `myPlotTemplateFile`.

```
lePlot("myPlotTemplateFile")
```

Sample lePlotOptions template file

```
lePlotOptions = '(nil
    area              "whole"
    view              "layout"
    cell              "Inv"
    library           "master"
    plot              "cellview"
    bBox              ((0.0 0.0) (10.0 36.0))
    noteText          "For Your Review"
    notes             t
```

```
    header            t
    plotterType       "postscript1"
    papersize         "A"
    plotter           "HP"
    plotToFile        nil
    display           "display"
    outputfile        ""
    time              "now"
    tmpdir            "/usr/tmp"
    copy              1
    scale             7408.333333
    center            nil
    mailto            "user_name"
    mail              t
    orientation       "automatic"
    plotsize          (2.916667 10.5)
    offset            (0.0 0.0)
    unit              "inches"
    gridMultiple      5
    gridSpacing       1.000000
    gridType          "Dots"
    stopLevel         32
    startLevel        0
    arrayDisplay      "Full"
    instName          "instance"
    pathCL            "yes"
    drawAxesOn        t
    iconsOn           nil
)
```

You can also insert the `lePlotOptions` into the `.cdsinit` file. Each time you print, the options from that template file are used. However, the library name, cell name, and the view name in the `lePlotOptions` template file are not used, they are generated by the cellview window from which you are printing.

Additional Information

The following table describes the various plot options.

Plot Option	Type	Description
`area`	String	`whole` plots entire cellview, `select` plots selected area, `reselect` plots a reselected area.
`arrayDisplay`	String	`Full` prints all instances in the array, `Border` prints only the instances around the outside edge of the array, `Source` prints only the instance in the lower left corner of the array.

Plot Option	Type	Description
`bBox`	Floating	Specifies in user units the part of the cellview to be plotted. Defaults to the cellview's bounding box.
`cell`	String	Name of the cell to be plotted. Used by `lePlot()` only.
`center`	Boolean	If `t`, the design is centered on the page.
`copy`	Integer	Number of copies to print.
`display`	String	Display name that specifies the display packets to use when generating the plot.
`drawAxesOn`	Boolean	If `t`, the axes are plotted.
`gridMultiple`	Integer	Major grid lines or dots are displayed every `gridMultiple` units. Major grid lines are thicker than minor grid lines. Major grid dots are larger than minor grid dots. This corresponds to the *Major Spacing* field in the Plot Display Options form.
`gridSpacing`	Integer	Minor grid lines or dots are displayed every `gridSpacing` units. Minor grid lines are thinner than major grid lines. Minor grid dots are smaller than major grid dots. This corresponds to the *Minor Spacing* field in the plot Display Options form.
`gridType`	String	Specifies grid type of `None`, `Dots`, or `Lines`.
`header`	Boolean	If `t`, displays a header and legend on the plot.
`iconsOn`	Boolean	If `t`, displays only outlines of instances in arrays.
`instName`	Boolean	If `t`, displays instance name.
`library`	String	Name of the cell's library to be plotted. Used by `lePlot()` only.
`mail`	Boolean	If `t`, plot notification is e-mailed to addresses specified by `mailto`. If `nil`, plot notification is not e-mailed.
`mailto`	String	Addresses to receive plot notification. Separate addresses with spaces.

Plot Option	Type	Description
`notes`	Boolean	If `t`, displays information specified by `noteText`.
`noteText`	String	Text displayed on the plot if `notes` is `t`.
`offset`	Floating	Specifies the X and Y origin of the cellview or viewing area in measurements specified by `unit`. The offset is a pair of floating-point values. If the plot spans more than one page, the offset is from the bottom left corner.
`orientation`	String	Sets what edge of the paper to use as the top. `portrait` plots the cellview as it appears in window, `landscape` rotates the cellview 90 degrees counterclockwise, `automatic` plots the cellview whichever way fits best.
`outputfile`	String	Saves plot information to the specified file instead of sending it to the printer.
`papersize`	String	Name of paper size for specified plotter.
`pathCL`	String	If `yes`, plots paths with center lines; if `no`, plots paths without centerlines; if `only`, plots centerlines only.
`plotsize`	Floating	Specifies the width and height of the resulting plot in inches. If `plotsize` is not specified, one of the following occurs: ■ If `scale` is specified, the `plotsize` is the `bBox` size scaled. ■ If area is `whole`, the `plotsize` is the size of the `bBox` for the plotter and `papersize`. ■ If none of the above is true, the `plotsize` defaults to 8x10.5 inches. Used by `lePlot()` only.
`plot`	String	Name of the cellview to be plotted.
`plotter`	String	Name of the plotter specified in the `.cdsplotinit` file.
`plotterType`	String	Specifies the type of file the printer expects.

Plot Option	Type	Description
`plotToFile`	Boolean	If `t`, the design is plotted to a file.
`scale`	Floating	Specifies the magnification of the plot in measurements specified by `unit`.
`startLevel`	Integer	Specifies the first level in the design hierarchy to be plotted.
`stopLevel`	Integer	Specifies the last level in the design hierarchy to be plotted.
`time`	String	Specifies the time to plot the design. The plot is stored in `tmpdir` until it is plotted. Use one of the following formats to set time: - `now` (plot immediately) - *`hh`*`:`*`mm`* `am/pm` *`day`* (plot at specified time, for example: `03:15 PM Wed`)
`tmpdir`	String	Specifies the directory where the plot is to be stored if it is not sent to the plotter.
`unit`	String	Specifies the type of measurement for `plotsize`, `scale`, and `offset`. Valid units are `inches`, `cm`, `mm`, and `feet`. Default is `inches`.
`view`	String	Name of the cell's view to be plotted. Used by `lePlot()` only.

Object Editing Functions

There are object editing functions that are procedural equivalents to the interactive editing functions on the *Edit* menu. The procedural functions perform like the interactive functions but have no user interface associated with them. You can use them to implement new user-defined functions.

Related Topics

leAlign

leAttachFig

leChopShape

leComputeAreaDensity

leAlign

```
leAlign(
     t_alignment
     )
     => t / nil
```

Description

Aligns the objects in the selection set. The objects are aligned to the left, right, top, or bottom edge of the selection box or along the center of the selection box in the vertical or horizontal direction.

Arguments

t_alignment	The way the objects need to be aligned. Valid Values: `left`, `right`, `top`, `bottom`, `vertical`, `horizontal`

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Aligns the selected objects to the left edge of the selection set.

```
leAlign("left")
```

leAttachFig

```
leAttachFig(
     d_figId1
     [ d_figId2 ]
     )
     => t / nil
```

Description

Attaches *d_figId1* to *d_figId2* by making *d_figId1* a child of *d_figId2*. If *d_figId1* is already attached to another object, it is detached from it. If *d_figId2* is not specified, *d_figId1* is detached from any object it is currently attached to without attaching it to anything else.

Arguments

d_figId1	The database ID of the child object.
d_figId2	The database ID of the parent object.

Value Returned

`t`	The child is attached.
`nil`	The child is not attached.

Example

Makes the object `fig1` a child of the object `fig2` and returns `t`:

```
leAttachFig( fig1 fig2 )
```

Detaches `fig1` from any object it is currently attached to without attaching it to anything else and returns `t`:

```
leAttachFig( fig1 )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts to point to the child object and the parent object. If you do not specify *w_windowId*, current window is used.

```
leHiAttach( [ w_windowId ] ) => t / nil
```

leChopShape

```
leChopShape(
     d_shapeId
     l_points
     g_closed
     [ g_remove ]
     [ x_sides ]
     )
     => l_newShapes / nil
```

Description

Cuts the shape *d_shapeId* using the chop shape *l_points*.

Arguments

d_shapeId	The database ID of the shape to be cut.
l_points	List of points describing the chop shape.
g_closed	Specifies whether the chop shape is closed or not. Valid Values: `t` or `nil`
g_remove	Specifies whether to remove the objects inside the chop shape. Default Value: `nil` Valid Values: `t` or `nil`
x_sides	Specifies the number of sides to be used when converting conic shapes to polygons before cutting them. Default Value: `20` Valid Values: any positive integer

Value Returned

l_newShapes	The list of shapes resulting from the operation if the shape is chopped.
`nil`	The shape is not chopped.

Example

Using the coordinates in the list, cuts a triangle out of the shape `shape2` after converting it to a polygon with 10 sides. Does not close the chop shape and does not remove any object. Returns the object IDs of the new shapes.

The objects are not removed if the chop shape is not closed. An open chop shape does not clearly indicate which objects are inside.

```
leChopShape( shape2 list(0:0 0:10 8:8) nil nil 10 )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the shape to be chopped, to create the chop shape, and specify if the shape is closed. If you do not specify *w_windowId*, the current window is used.

```
leHiChop( [ w_windowId ] ) => t / nil
```

leComputeAreaDensity

```
leComputeAreaDensity(
     w_windowId
     l_lppSpec
     [ ?depth x_depth ]
     [ ?region l_region ]
     )
     => l_result / nil
```

Description

Computes the total area and the density of shapes with specified layers within a region. If part of the shape is inside the region, only the area of the part that is inside the region is calculated.

Arguments

w_windowID	Window ID of the window used for computing area and density.
l_lppSpec	The list of layer-purpose pairs to be considered for area and density calculation. You can specify the layers in any of the following formats: ■ One layer: `list(list(list("Met1" "drawing")))` ■ Multiple layers: `list(list(list("Met1" "drawing1")) list(list("Met2" "drawing")) list(list("Met3" "drawing1")))` ■ Merge layers: `list(list(list("Met1" "drawing") list("M1" "draw"))` ■ Multiple merge layers: `list(list(list("Met1" "drawing") list("M1" "drawing")) list(list("Met2" "drawing") list("M2" "drawing")))`
`?depth` *x_depth*	The depth of hierarchy to be used when calculating the area and density. Valid Values: any integer from `0` to `31`. Defaults to `0`.
`?region` *l_region*	The region point list for which the area and density of shapes needs to be calculated. If part of a shape is inside the region, then only the area of the part that is inside the region is calculated.

Value Returned

l_result	List of DPL. For each merged layer, DPL is (`nil density densityValue area areaValue`)
`nil`	The function was not successful.

Example

```
leComputeAreaDensity(hiGetCurrentWindow() list(list(list("Nwell" "drawing")))
?depth 31 ?region list(-0.394:-0.125 1.629:-0.125 1.629:1.355 -0.394:1.355))
```

leComputeFigArea

```
leComputeFigArea(
     d_figId
     )
     => n_area / nil
```

Description

Computes the area of an object. The following objects are supported: rectangle, label, text display, polygon, ellipse, dot, marker, donut, path, figGroup, row, pathSeg, transmission line (trl), bend, taper, area or layer blockage, and area, snap, cluster, or place and route (PR) boundary.

Arguments

d_figId	Database ID of the object for which the area is computed.

Value Returned

n_area	The area of the object.
nil	The area of the object is not computed.

Example

Returns `30.0` as the area of the rectangle.

```
cv = geGetEditCellView()
rect = dbCreateRect(cv "poly" '((2 2) (8 7)))
leComputeFigArea(rect)
30.0
```

Related Topics

Object Editing in Layout XL

leConvertInstToMosaic

```
leConvertInstToMosaic(
     d_instId | ld_instId
     [ mode { t | nil } ]
     )
     => ld_mosaicId / nil
```

Description

Converts one or more selected instances into one or more mosaics.

Arguments

d_instId	Database ID of the instance to be converted.
ld_instId	List of database IDs of the instances to be converted.
mode	Specifies whether the command creates a number of individual mosaics or a single mosaic. The default is `t`, which means that each of the instances specified is converted into an individual mosaic. When set to `nil`, the instances passed in are converted into a single mosaic provided they are placed in an appropriate matrix.

Value Returned

ld_mosaics	List of database IDs of the mosaics created.
`nil`	The conversion failed.

Examples

Converts `inst1` into a mosaic:

```
leConvertInstToMosaic(inst1)
```

Converts `inst1` and `inst2` into two mosaics:

```
leConvertInstToMosaic(list(inst1, inst2))
```

Converts instances into a single mosaic:

```
leConvertInstToMosaic(list(instId1, instId2) nil)
```

leConvertMosaicToModgen

```
leConvertMosaicToModgen(
     d_mosaicId | ld_mosaicId
     )
     => d_modgenId | ld_modgenId / nil
```

Description

Converts one or more selected mosaics into one or more modgens.

Arguments

d_mosaicId	Database ID of the mosaic to be converted.
ld_mosaicId	List of database IDs of the mosaic to be converted.

Value Returned

d_modgenId	Database ID of the mosaic created.
ld_modgenId	List of database IDs of the mosaics created.
`nil`	The conversion failed.

Examples

Select one or more mosaics on the canvas then run command:

```
leConvertMosaicToModgen(geGetSelSet())
```

The function returns the list of database IDs of all converted modgens. In case, no modgen is converted, the function returns `nil`.

leConvertPolygonToPath

```
leConvertPolygonToPath(
     d_polygonId
     )
     => d_pathId / nil
```

Description

Converts the polygon *d_polygonId* to a path. If the conversion is successful, the polygon is replaced by a path with the same boundary as the polygon.

Arguments

d_polygonId	The database ID of the polygon to convert.

Value Returned

d_pathId	The object ID of the path if it is created.
nil	The path is not created.

Example

```
when( pathId = leConvertPolygnToPath(polyId)
  DoSomething(pathId)
  )
```

leConvertSelectedDynamicShapes

```
leConvertSelectedDynamicShapes(
     d_cellViewId
     )
     => t / nil
```

Description

(Virtuoso RF Option) Converts the selected dynamic shapes to static shapes.

Arguments

d_cellViewId	The database ID of the cellview that has dynamic shapes to be converted to static shapes.

Value Returned

t	The selected dynamic shapes are converted to static shapes.
nil	The selected dynamic shapes are not converted to static shapes.

Example

Cellview that has dynamic shapes to be converted to static shapes.

```
win = hiGetCurrentWindow()
cellView = geGetEditCellView(win)
leConvertSelectedDynamicShapes(cellView)
```

leConvertShapeToPathSeg

```
leConvertShapeToPathSeg(
     d_shapeId
     )
     => l_newShapes / nil
```

Description

Converts a shape to an equivalent pathSeg. The function follows the routing direction defined for the corresponding layer in the technology file.

Arguments

d_shapeId	The database ID of the shape to convert.

Value Returned

l_newShapes	The list of shapes resulting from the function if the shape is converted to pathSegs.
nil	The shape is not converted to a pathSeg.

Example

Converts the shape `shape2` to a pathSeg and returns a list of object IDs of the new shapes.

```
leConvertShapeToPathSeg(shape2)
```

leConvertShapeToPolygon

```
leConvertShapeToPolygon(
     d_shapeId
     [ x_sides ]
     )
     => d_polygonId / nil
```

Description

Converts shape *d_shapeId* to an equivalent polygon. If it is a conic, the polygon has the specified number of sides.

Arguments

d_shapeId	The database ID of the shape to convert.
x_sides	Specifies the number of sides to use when converting conic shapes. Default Value: `20` Valid Values: any positive integer up to `2047`.

Value Returned

d_polygonId	The object ID of the polygon if it is created.
`nil`	The polygon is not created.

Example

Converts the conic shape `shape2` to a polygon with 10 sides and returns the object ID for the polygon.

```
leConvertShapeToPolygon( shape2 10 )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the shape to convert. If you do not specify *w_windowId*, the current window is used.

```
leHiConvertShapeToPolygon( [ w_windowId ] ) => d_polygonId / nil
```

leConvertTrimmedShapesToPRStyle

```
leConvertTrimmedShapesToPRStyle(
     d_cvId
     [ ?layerName t_layerName ]
     [ ?depth x_depth ]
     [ ?library l_library ]
     )
     => x_count / nil
```

Description

Converts pathSegs that are trimmed by trim shapes in custom style to PR style so that they are readable in Innovus. Only pathSegs trimmed by trim shapes in the same cellview are converted. PathSegs that are trimmed by trim shapes in a different clone or in a different cellview are not converted. The function searches all trimmed pathSegs on the specified layer in the hierarchy up to the specified depth and converts them to PR style only in the cellviews of the libraries specified. The function also adds the relevant patch type to pathSegs abutted to trim shapes.

The function needs edit permissions for the top-level cellview and libraries specified in the hierarchy.

Arguments

d_cvId	Database ID of the cellview in which pathSegs are to be converted.
?layerName *t_layerName*	Layer name on which pathSegs are to be converted.
?depth *x_depth*	Depth up to which the design hierarchy is to be traversed to find pathSegs and trim shapes.
?library *l_library*	Names of libraries from which cellviews in the design hierarchy are to be converted.

Value Returned

x_count	The number of pathSegs converted.
nil	The conversion failed.

Examples

Converts 12 custom style pathSegs to PR style pathSegs on Metal1 layer in the design hierarchy up to level 5 in the cellviews belonging to lib1 or lib2 libraries.

```
leConvertTrimmedShapesToPRStyle(geGetEditCellView() ?depth 5 ?layerName "Metal1"
?library "lib1 lib2")
12
```

leCopyTemplatesToCellView

```
leCopyTemplatesToCellView(
     d_cellViewId
     )
     => t / nil
```

Description

Copies the row region templates and WSPDefs from the source cellview specified with the environment variable autoCopyTemplateCellViewList to the cellview specified in this function. If the cellview has existing templates, the new templates from the source cellview are not copied.

Arguments

d_cellViewId	The database ID of the cellview to which the row region templates and WSPDefs are to be copied.

Value Returned

t	The row region templates and WSPDefs have been copied.
nil	The row region templates and WSPDefs have not been copied.

Example

```
envSetVal("layout" "autoCopyTemplateCellViewList" 'string "(sourceLibName
sourceDesignName layout)")
cvId = geGetEditCellView()
```

Cellview where templates are to be copied.

```
leCopyTemplatesToCellView(cvId)
```

leCreateLayerField

```
leCreateLayerField(
     s_name
     ?callback t_callback
     ?appendChoices l_appendChoices
     ?prependChoices l_prependChoices
     ?maxVisibleItems x_maxVisibleItems
     ?syncedPaletteWindow w_syncedPaletteWindow
     ?techFile d_techFile
     ?value t_value
     ?enabled g_enabled
     ?invisible g_invisible
     ?layerFunction t_layerFunction
     ?maxWidth x_maxWidth
     )
     => t_layerfield
```

Description

Creates a cyclic field for to display layer-purpose pair (LPP) names with swatches. The combo box created allows you to type and search for a layer purpose pair within the list of values. The list of layers is displayed only if a layout window is open.

Arguments

s_name
: A symbol specifying the name the field will be accessed in the form.

?callback *t_callback*
: The name of a function or SKILL code. If a single name without '()' is specified, for example, "`metal1 drawing`", the current value is passed as an argument when the value changes.

?appendChoices *l_appendChoices*
: A list of strings to be appended to the LPPs. If the strings are LPPs, they are displayed with the associated swatches similar to LPPs. LPPs specified through these are options are not filtered through `layerFunction` or `syncedPaletteWindow`.

?prependChoices *l_prependChoices*

A list of strings that should be prepended to the LPPs. For example, `"Auto select"` and `"Use Current Entry Layer"`.

`?maxVisibleItems` *x_maxVisibleItems*

The number of entries that will be visible when you click the drop down list. Default value: `10`.

`?syncedPaletteWindow` *w_syncedPaletteWindow*

The window to synchronize the layer purpose list to. This is effective only when the `validLppFilterOn` is on.

`?techFile` *d_techFile*

The technology file from which the valid layer purpose needs to be read.

`?value` *t_value* The current value of the field. You can set or get this value.

`?enabled` *g_enabled*

Specifies whether the layer field is enabled.

`?invisible` *g_invisible*

Specifies whether the layer field is visible.

`?layerFunction` *t_layerFunction*

The name of the function that is used to filter out the LPPs to be shown in layer field. A valid layer function filters the LPPs found in the given `techField`. If you need a layer field with a specific set of layers that do not get filtered by Palette settings, specify the name of a non-existent layer function, such as `Dummy1`. The only choices then available are those specified by the `appendChoices` or `prependChoices` arguments.

`?maxWidth` *x_maxWidth*

The maximum width of the layer field.

Value Returned

t_layerfield	The name of the created layer field.

Example

```
leCreateLayerField( 'moveToLayer ?callback "layerFieldCB" ?techFile
techGetTechFile(geGetEditCellView()) ?syncedPaletteWindow hiGetCurrentWindow() )
```

leCreateNetField

```
leCreateNetField
     name
     ?value t_value
     ?callback t_callback
     x_maxVisibleItems
     )
     => netfield
```

Description

Creates cyclic field for showing net names of the current cellview. The combo box created allows you to type in and search for the net names within the list of values.

Arguments

name	A symbol specifying the name the field will be accessed in the form.
?value *t_value*	The current value of the field. You can set or get this value.
?callback *t_callback*	The name of a function or SKILL code.
x_maxVisibleItems	The number of entries that will be visible when you click the drop down list.

Value Returned

netfield	The created net field.

Example

```
leCreateNetField('testNetField ?window hiGetCurrentWindow() ?callback "myTestCB")
```

leCycleSnapModes

```
leCycleSnapModes(
     )
     => t
```

Description

Changes both the edit and create snap modes, in the order of `diagonal`, `anyAngle`, and `orthogonal`. If the edit and create mode values are different (unsynchronized), the first value for Create/Edit snap mode will be `diagonal`.

Arguments

None

Value Returned

`t`	The command executed successfully.

Example

```
leCycleSnapModes()
```

If the Create/Edit snap mode is currently set to `diagonal`, the Create/Edit snap mode value will be cycled as follows:

```
leCycleSnapModes() -> "anyAngle"
leCycleSnapModes() -> "orthogonal"
leCycleSnapModes() -> "diagonal"
```

leDecrementStopLevelByOne

```
leDecrementStopLevelByOne(
     w_windowId
     [ t_useCellViewMaxDepth ]
     )
     => t / nil
```

Description

Decrements the stop level value of the specified window by one when it is greater than zero.

Arguments

w_windowId	Database ID for the window.
t_useCellViewMaxDepth	When *t_useCellViewMaxDepth* is set to `t`, the stop level is decremented using the minimum value between maximum cellview depth (opened in memory) and window stop level. For example, if the window stop level is `31` and the maximum cellview depth is `8`, this function sets the stop level to `7`. Valid Values: `t` or `nil`. The default is `nil`.

Value Returned

`t`	The stop level was decremented by one.
`nil`	The stop level was not decremented by one.

Example

Decrements the stop level value of the current window by one.

```
leDecrementStopLevelByOne(hiGetCurrentWindow)
```

leDefineExternalPins

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

Description

Starts the Define External Pins enter function in the current window. The Define External Pins enter function provides a user interface for defining one or more pins on a net as being externally connected outside of a device.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Starts the Define External Pins enter function. Prints the message when the command successfully finishes.

```
when(leDefineExternalPins()
        info("Define External Pins executed successfully.\n")
)
```

Related Topics

Pin Creation

leDefineInternalPins

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

Description

Starts the Define Internal Pins enter function in the current window. The Define Internal Pins enter function provides a user interface for defining one or more pins on a net as being internally connected within a device.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Starts the Define Internal Pins enter function. Prints the message when the command successfully finishes.

```
when(leDefineInternalPins()
        info("Define Internal Pins executed successfully.\n")
)
```

leDefinePPPins

```
leDefinePPPins(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the Define Pseudo Parallel Connected Net enter function in the current window. Corresponds to the Connectivity – Pins – Pseudo Parallel Connect command. The Define Pseudo Parallel Connected Net form provides a user interface for defining one or more pins on a net as being pseudo parallel connected. If you do not specify *w_windowId*, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

Example

Starts the Define Pseudo Parallel Connected Net enter function. Prints the message when the command successfully finishes.

```
when(leDefinePPPins()
        info("Define Pseudo Parallel Connected Net executed successfully.\n")
)
```

leDefineWeaklyConnectedPins

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

Description

Starts the Define Weak Pins enter function in the current window. The Define Weak Pins enter function provides a user interface for defining one or more pins on a net as being weakly connected within a device.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Starts the Define Weak Pins enter function. Prints the message when the command successfully finishes.

```
when(leDefineWeaklyConnectedPins()
        info("Define Weakly Connected Pins executed successfully.\n")
)
```

leEnableInstPropEditFields

```
leEnableInstPropEditFields(
     t_libName
     [ ?libField g_libField ]
     [ ?cellField g_cellField ]
     => t / nil
```

Description

Enables or disables the *Library* or *Cell* fields for an instance in the Property Editor form.

Arguments

t_libName	Specifies the name of the library for which the *Library* or *Cell* fields are to be enabled or disabled. Valid Values: any valid library name
?libField *g_libField*	An optional boolean value specifying whether the *Library* field is enabled or not. The default is `t`.
?cellField *g_cellField*	An optional boolean value specifying whether the *Cell* field is enabled or not. The default is `t`.

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

To enable the *Library* and *Cell* fields for "`testlib1`" library for an instance in the Property Editor form, specify:

```
leEnableInstPropEditFields("testlib1" ?libField t ?cellField t)
```

To disable the *Library* and *Cell* fields for "`testlib1`" library for an instance in the Property Editor for,. specify:

```
leEnableInstPropEditFields("testlib1" ?libField nil ?cellField nil)
```

leFlattenInst

```
leFlattenInst(
     d_instId
     x_levels
     [ g_flattenPCells ]
     [ g_preservePins ]
     [ g_preserveRODobjs ]
     [ g_delDetachedBlockages ]
     [ g_preservePinFigs ]
     [ g_flattenVias ]
     [ g_preserveTermName ]
     [ t_excludePcellListFileName ]
     )
     => t / nil
```

Description

Flattens instance *d_instId* up through *x_levels* of hierarchy. The optional arguments are positional: you must specify `nil` for optional arguments prior to an optional argument you want to specify. For example, if you do not want to specify *g_flattenPCells*, but want to set *g_preservePins* to `t`, you need to specify both (`nil t`).

In color-aware designs, effective coloring on both locked and unlocked shapes is preserved. However, if the coloring engine is enabled during flattening, unlocked shapes might be recolored after flattening is completed.

Arguments

d_instId	The database ID of the instance to flatten.
x_levels	Number of levels of hierarchy through which to flatten the instance. Valid Values: an integer between `0` and `32`
g_flattenPCells	Specifies whether parameterized cells are flattened. Default Value: `nil` Valid Values: `t` or `nil`
g_preservePins	Specifies whether pin connectivity information is preserved. Default Value: `nil` Valid Values: `t` or `nil`

g_preserveRODobjs Specifies whether ROD properties are preserved.

Default Value: `nil`

Valid Values: `t` or `nil`

g_delDetachedBlockages

Specifies whether detached blockages are preserved.

Default Value: `nil`

Valid Values: `t` or `nil`

g_preservePinFigs Specifies whether geometric information of the flatten pins is preserved.

Default Value: `t`

Valid Values: `t` or `nil`

g_flattenVias Specifies whether vias are flattened.

Default Value: `t`

Valid Values: `t` or `nil`

g_preserveTermName

Specifies whether terminal name of the flatten pins is preserved.

Default Value: `nil`

Valid Values: `t` or `nil`

t_excludePcellListFileName

Specifies the name of the file that contains the list of Pcells that should not be flattened.

Default Value: `""`

Value Returned

`t`	The instance is flattened.
`nil`	The instance is not flattened.

Example

Flattens the instance `instance1` through two levels of the hierarchy but does not flatten any parameterized cells contained in the instance.

```
leFlattenInst( instance1 2 nil )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the instance to flatten and specify the number of levels through which to flatten it. If you do not specify *w_windowId*, the current window is used.

```
leHiFlatten( [ w_windowId ] ) => t / nil
```

leFreezeInst

```
leFreezeInst(
     d_instId
     t_libName
     t_cellName
     t_viewName
     [ g_overwrite ]
     [ g_reuse ]
     )
     => t / nil
```

Description

Converts a Pcell instance to a non-Pcell instance of a new master created on the disk for the specified library, cell, and view. This function is supported for mosaics.

Arguments

d_instId	The database ID of the Pcell instance to be converted to a non-Pcell instance.
t_libName	Specifies the library name of the new cellview. Valid Values: any valid library name
t_cellName	Specifies the name of the new cell. Valid Values: any valid cellview name
t_viewName	Specifies the view name of the new cell. Valid Values: any valid view name
g_overwrite	Overwrites the existing master, if set to `t`.
g_reuse	Reuses the existing frozen Pcell master, if available, when set to `t`. The default value is `nil`.

Value Returned

`t`	The Pcell instance is converted to a non-Pcell instance.
`nil`	The Pcell instance is not converted to a non-Pcell instance.

Example

Converts the Pcell instance `inst1` to a non-Pcell instance in the view `view` of the cell `cell2` in the library `lib`. If a frozen Pcell master already exists, it is reused.

```
leFreezeInst(inst1 "lib" "cell2" "view" nil t)
```

Related Topics

Freeze Pcell Instance Form

dbFreeze

leHiAreaDensity

```
leHiAreaDensity(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Area and Density Calculator form in the specified window. This form lets you compute area and density of shapes. You can choose options to specify region, LPP, and hierarchical depth for the shapes to be considered for area and density calculations. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

leHiAssignNet

```
leHiAssignNet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the Assign Net enter function that provides the user interface for assigning new and existing nets to shapes, instance pins, vias, and pathSegs.

For instance pins this functionality works only in Layout XL and higher tiers.

The function supports connectivity propagation to physically connected shapes, instance pins, vias, and pathSegs.

Arguments

w_windowId	ID of the window containing the objects for which nets are to be assigned. If not specified, the command operates on the objects in the currently active layout window.

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

Examples

The following example starts the command to assign nets to the objects in the currently active layout window.

```
leHiAssignNet()
```

leHiConvertMosaicToInst

```
leHiConvertMosaicToInst(
     [ w_windowId ]
     )
     => t / nil
```

Description

Converts mosaics to instances.

Arguments

w_windowID	Window ID of the window.

Value Returned

t	The conversion was successful.
nil	The conversion was not successful.

Examples

```
leHiConvertMosaicToInst()
```

leHiConvertMosaicToModgen

```
leHiConvertMosaicToModgen(
     [ w_windowId ]
     )
     => t / nil
```

Description

Converts one or more selected mosaics to an equivalent number of modgens. If *w_windowId* is not specified, the current window is used. You are prompted to select one or more mosaics to convert into modgens.

Arguments

w_windowID	Window ID of the window.

Value Returned

t	The conversion was successful.
nil	The conversion was not successful.

Examples

The `leHiConvertMosaicToModgen` can run in the pre-selection and post-selection mode. In the pre-selection mode, select one or more mosaics on the canvas then run following function:

```
leHiConvertMosaicToModgen()
```

The command ends after conversion.

In the post-selection mode, you need to run the `leHiConvertMosaicToModgen()` and then, select shapes which are to be converted to modgen. The command continues after modgen conversion.

leHiEditSlot

```
leHiEditSlot(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the *Edit Slot* form in the specified window. You are prompted to select a rectangle or path containing slots, the settings of which are used to populate the *Edit Slot* form. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

leHiUnassignNet

```
leHiAssignNet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Unassigns nets to shapes, instance pins, and routed objects like vias and pathSegs.

Note: For instance pins this functionality works only in Layout XL and higher tiers.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

Examples

```
leHiUnassignNet()
```

leIncrementStopLevelByOne

```
leIncrementStopLevelByOne(
     w_windowId
     )
     => t / nil
```

Description

Increments the stop level value of the specified window by one.

Arguments

w_windowId	Database ID for the window where you want to run the function. The current window is used if you do not specify a window ID.

Value Returned

`t`	The stop level was incremented by one.
`nil`	The stop level was not incremented.

Example

Increments the stop level value of the current window by one.

```
leIncrementStopLevelByOne(hiGetCurrentWindow)
```

leIOZoomToFigGroupTopLevelContext

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

Description

Zooms to the location of the selected figGroup occurrence in the main window.

This function works only when the instance occurrence overlap view is generated in the same Virtuoso session.

Arguments

None

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

```
leIOZoomToFigGroupTopLevelContext()
```

leLmbCtrlOption

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

Description

Allows some Virtuoso Layout Suite creating and editing commands to override the `Ctrl + Left` mouse click. The command options are embedded in the Virtuoso Layout Suite software and cannot be changed.

Arguments

None

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

```
leLmbCtrlOption()
```

leLmbShiftOption

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

Description

Allows some Virtuoso Layout Suite creating and editing commands to override the `Shift + Left` mouse click. The command options are embedded in the Virtuoso Layout Suite software and cannot be changed.

Arguments

None

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

```
leLmbShiftOption()
```

leMakeCell

```
leMakeCell(
     l_figs
     t_libName
     t_cellName
     t_viewName
     g_replace
     [ ?preserveConn g_preserveConn ]
     [ ?createPins g_createPins ]
     [ ?transInst g_transInst ]
     [ ?pinsBelowBoundary g_pinsBelowBoundary ]
     [ l_point ]
     [ t_cellType ]
     )
     => d_cellViewId / nil
```

Description

Creates the cellview specified by *t_libName*, *t_cellName*, and *t_viewName*. If the cellview already exists, it is overwritten. The arguments ?preserveConn, ?createPins, ?transInst, and ?pinsBelowBoundary are only supported only in Layout XL and higher tiers.

Arguments

l_figs	List of objects contained in the original cellview.
t_libName	Specifies the name of the new cellview's library. Valid Values: any valid library name
t_cellName	Specifies the name of the new cellview's cell. Valid Values: any valid cellview name
t_viewName	Specifies the name of the new cellview's view. Valid Values: any valid view name
g_replace	Specifies whether the objects selected should be replaced by an instance of the new cellview. If this option is not selected, the original objects are not modified. Valid Values: t or nil
?preserveConn *g_preserveConn*	Controls whether the made cell has its connectivity preserved.
?createPins *g_createPins*	

Controls whether the made cell has its pins created.

`?transInst` *`g_transInst`*

Controls whether the made cell occurrence is transparent.

`?pinsBelowBoundary` *`g_pinsBelowBoundary`*

Specifies whether the pins of the made cell are placed outside the design boundary. Valid values: `t` or `nil`.

This argument is only available if `?preserveConn` and `?createPins` are set.

`l_point`	Specifies the new origin for the cellview. Valid Values: list of two, an X and a Y coordinate
`t_cellType`	Specifies the cell type for the cellview. Valid Values: `none`, `block`, `blockRing`, `cover`, `coverBump`, `pad`, `padSpacer`, `core`, `coreSpacer`, `coreAntenna`, `corner`, `softMacro`, `via`, `blockBlackBox`, `padArea`

Value Returned

`d_cellViewId`	The new cellview ID if a new cellview is created.
`nil`	The new cellview is not created.

Example

Copies of the objects `fig1` and `fig2` are created in a new cellview `mycell`. The objects in the original cellview are not replaced by an instance of the cellview. Returns the database ID of the new cellview.

```
leMakeCell( list(fig1 fig2) "mylib" "mycell" "layout" nil )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the objects contained in the new cellview; specify the library, cell, and view names; specify whether to replace the selected objects; and indicate an origin for the new cellview. If you do not specify *`w_windowId`*, the current window is used.

```
leHiMakeCell( [ w_windowId ] ) => t / nil
```

```
leMakeCell( list(figId1 figId2) "design" "top" "layout" t list(0 0) "softMacro"
?preserveConn t ?createPins t ?transInst t )
```

Copies of the objects `figId1` and `figId2` are created in a new cellview `top` in the `design` library. The made cellview has a transparent instance created that has its connectivity preserved and pins created.

```
leMakeCell( list(figId3 figId4) "design" "mycell2" "layout" t list(0 0) "softMacro"
?preserveConn t ?createPins t ?transInst t ?pinsBelowBoundary t)
```

Copies of the objects `figId3` and `figId4` are created in a new cellview `mycell2` in the `design` library. The made cellview has a transparent instance created that has its connectivity preserved across hierarchies and its pins are placed outside the design boundary.

leMakeHierReadonlyOrEditableMC

```
leMakeHierReadonlyOrEditableMC(
     [ w_windowId ]
     )
     => t / nil
```

Description

Makes a cellview either read-only or editable. When one or more instances are selected before invoking this function, only the selected instance(s) are shown in the browser. When no instances are selected before invoking this function, all instances in the cellview are shown in the browser. If you do not specify *w_windowId*, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

leMergeShapes

```
leMergeShapes(
     l_shapes
     [ x_sides ]
     )
     => l_newShapes / nil
```

Description

Merges the list of shapes *l_shapes*. The input shapes can be on multiple layers. Merges only shapes on the same layer. Returns the list of shapes resulting from the operation.

Arguments

l_shapes	List of shapes to be merged.
x_sides	Specifies the number of sides to use when converting conic shapes to polygons before merging them. Default Value: `20` Valid Values: any positive integer

Value Returned

l_newShapes	The list of shapes resulting from the merge.
`nil`	The shapes did not merge.

Example

Merges the objects `fig1`, `fig2`, and `fig3` after converting any conics to polygons with the default of 20 sides and returns the object ID of the resulting shape.

```
leMergeShapes( list(fig1 fig2 fig3) )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the shapes to be merged. If you do not specify *w_windowId*, the current window is used.

```
leHiMerge( [ w_windowId ] ) => t / nil
```

leMicroEdit

```
leMicroEdit(
     t_direction
     [ w_windowId ]
     )
     => t / nil
```

Description

Allows using bindkeys instead of the mouse to move, copy, or stretch any shape. When no command is active, the selected shape is either moved or stretched, depending on whether the shape is fully or partially selected, respectively. When the copy, move, or stretch command is active, the selected shape is copied, moved, or stretched, respectively.

By default the length of mouse movement for each keystroke is 0.5 user units. You can change the length by setting the `legRpDelta` variable in CIW, such as `legRpDelta = 1` or `legRpDelta = 2`.

Arguments

t_direction	The direction of the command. The choices are `up`, `upperRight`, `left`, `lowerLeft`, `down`, `lowerRight`, `right`, or `upperLeft`.
w_windowId	Database ID of the window.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

Example

In the following example, the numerical pad bindkeys have been set for each direction of the command.

```
hiSetBindKey( "Layout" "<Key>KP_7" "leMicroEdit('upperLeft)")
hiSetBindKey( "Layout" "<Key>KP_4" "leMicroEdit('left)")
hiSetBindKey( "Layout" "<Key>KP_1" "leMicroEdit('lowerLeft)")
hiSetBindKey( "Layout" "<Key>KP_2" "leMicroEdit('down)")
hiSetBindKey( "Layout" "<Key>KP_3" "leMicroEdit('lowerRight)")
```

```
hiSetBindKey( "Layout" "<Key>KP_6" "leMicroEdit('right)")
hiSetBindKey( "Layout" "<Key>KP_9" "leMicroEdit('upperRight)")
hiSetBindKey( "Layout" "<Key>KP_8" "leMicroEdit('up)")
```

leModifyCorner

```
leModifyCorner(
     d_figId
     l_selArray
     g_chamfer
     x_distance
     [ x_sides ]
     )
     => t / nil
```

Description

Rounds or cuts off selected corners of the rectangle or polygon specified by *d_figID*.

Arguments

d_figId	The database ID of the polygon or rectangle to modify.
l_selArray	A list of Boolean values indicating whether each vertex in the shape can be modified. The Boolean values must match the number and order of vertices in the shape. (Use *d_shapeId*~>Points to obtain the vertices of the shape.)
g_chamfer	A Boolean value specifying whether you want to create a chamfer or a rounded corner. Valid Value: t (chamfer) or nil (round) Valid Values: any positive floating value up to half the length of the longest adjacent line segment
x_distance	The distance from the vertex to begin modifying the corner. The distance used is half the length of the shortest line segment adjacent to the corner or the value supplied for *x_distance*, whichever is shorter.
x_sides	The number of sides to use when converting a corner to a curve. Default Value: 20 Valid Values: any positive integer

Value Returned

`t`	The corner is modified.
`nil`	The corner is not modified.

Example

```
leModifyCorner( figId list(t nil nil nil nil nil) nil 1.0)
```

where

figId~>points are:

```
((10.15 5.77)
(13.49 5.77)
(13.49 1.39)
(16.92 1.39)
(16.92 -0.48)
(10.15 -0.48)
)
```

In the example, the second argument is a list of six booleans. The vertex for which you have specified `t` in the second argument will be converted to chamfers.

Interactive Function

Enter this function with only the window ID argument; the system prompts you to select the corners to modify. If you do not specify *`w_windowId`*, the current window is used.

```
leHiModifyCorner( [ w_windowId ] ) => t / nil
```

leMoveCellViewOrigin

```
leMoveCellViewOrigin(
     d_cellViewId
     l_point
     )
     => t / nil
```

Description

Moves the contents of cellview *d_cellViewId* so that the origin is at *l_point*.

Arguments

d_cellViewId	Database ID for the cellview whose origin is moved.
l_point	Specifies the new origin for the cellview.

Value Returned

t	The origin moves.
nil	The origin does not move.

Example

Moves the contents of cellview `cell2` a negative 14 user units in the X direction and a negative 14 user units in the Y direction, so that the origin is at the original `14:14` location. Returns `t`.

```
leMoveCellViewOrigin( cell2 14:14 )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the new origin. If you do not specify *w_windowId*, the current window is used.

```
leHiMoveOrigin( [ w_windowId ] ) => t / nil
```

lePasteFigs

```
lePasteFigs(
     d_cellViewId
     l_destPt
     [ t_rotation ]
     [ t_referenceParameter ]
     )
     => t / nil
```

Description

Places the contents of the yank buffer in cellview *d_cellViewId* at the destination point *l_destPt*.

Arguments

d_cellViewId	Database ID for the cellview in which to place the yanked objects.
l_destPt	Coordinate specified as the destination point for the reference point of the yanked objects.
t_rotation	Rotates the contact clockwise, and mirrors the contact. Default Value: `R0` Valid Values: `R0, R90, R180, R270, MX, MXR90, MY,` or `MYR90`
t_referenceParameter	Reference point of the bounding box. Default Value: `user specified` Valid Values: `lowerLeft`, `centerLeft`, `upperLeft`, `lowerCenter`, `centerCenter`, `upperCenter`, `lowerRight`, `centerRight`, `upperRight`, `origin`, or `user specified`.

Value Returned

`t`	The objects are placed.
`nil`	The objects are not placed.

Example

Pastes the objects currently in the yank buffer, placing the reference point of the yanked objects at 100:100.

```
lePasteFigs( geGetEditCellView() 100:100 )
```

Interactive Function

Enter this function with the window ID argument only; the system prompts you to point to the location to place the yanked objects. If you do not specify *w_windowId*, the current window is used.

```
leHiPaste( [ w_windowId ] ) => t / nil
```

Related Topics

leYankFigs

lePlGetApplyPrevData

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

Description

Prints the previously applied data from the property editor cache of the current session.

Arguments

None

Value Returned

`t`	Previously applied properties data is listed, showing for each modified tab the field names and their last-applied values.
`nil`	No previously applied properties data is listed because no changes were made in the current session.

Example

Prints the last-applied data from the current session, showing the values that were modified in the *Width* field of the *Attributes* tab and the *Net Name* field of the *Connectivity* tab of the property editor.

```
lePlGetApplyPrevData()
Previously applied properties data
    Attribute
        Width : 1.910000
    Connectivity
        Net Name : VSS
t
```

Related Topics

Editing Properties of an Object

lePlGetEditorName

```
lePlGetEditorName(
     [ w_windowId ]
     )
     => t_editorname
```

Description

Returns the editor name of the Edit Properties form if the form is open in the specified window.

Arguments

w_windowId	ID of the window where you want to run the function. The current window is used if you do not specify a window ID.

Value Returned

t_editorname	Editor name of the Edit Properties form if the form is open. Otherwise, the function returns an empty string.

Example

Returns `editorNum0` as the editor name of the Edit Properties form open in the current window.

```
lePlGetEditorName()
editorNum0
```

leQuickAlignToggleMode

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

Description

Allows you to toggle between quick align modes - move or stretch, and copy.

Arguments

None

Value Returned

`t`	Quick align mode is toggled.
`nil`	Quick align mode is not toggled.

Example

Toggles the quick align mode to move or stretch if it was previously copy, or copy if it was previously move or stretch.

```
leQuickAlignToggleMode()
```

leReturn

```
leReturn(
     [ w_windowId ]
     )
     => t / nil
```

Description

Applies to instances, nested instances, groups, and nested groups for which you are editing-in-place. When editing instances or nested instances in Edit-in-Place mode, returns one level; when there is only one level, exits Edit-in-Place mode. When editing a un-nested group, exits Edit-in-Place Mode; when editing a nested group, returns to the group of which the edited group is a member. If you do not specify *w_windowId*, the current window is used.

Arguments

w_windowId	Database ID for the window in which to return during Edit-In-Place mode.

Value Returned

t	The return during Edit-in-Place mode was successful.
nil	The return during Edit-in-Place mode was not successful.

Example

Returns the window specified by *currWindowId* from the Edit-In-Place mode.

```
leReturn(currWindowId)
```

leReturnToLevel

```
leReturnToLevel(
     [ w_windowId ]
     )
     => t / nil
```

Description

Applies to instances, nested instances, groups, and nested groups for which you are editing-in-place. For instances, returns one or more levels up the hierarchy from the current level in the instance you last edited-in-place. When there is only one level, exits Edit-in-Place mode. When you have descended more than one level, this function displays the Return To Level form, showing the levels to which you can return. You select (highlight) the desired level and click *OK*. For example, if you are using `leHiEditInPlace` to edit a cellview two levels deep in your hierarchy, you can use `leReturnToLevel` to move up one or two levels in the hierarchy. For groups, returns from the group, and if the group is nested, from all groups in the nesting, and exits Edit-in-Place mode. If you do not specify *`w_windowId`*, the current window is used.

Arguments

`w_windowId`	Database ID for the window in which to return during Edit-In-Place mode.

Value Returned

`t`	The return during Edit-in-Place mode was successful.
`nil`	The return during Edit-in-Place mode was not successful.

Example

Returns the window specified by *`currWindowId`* from the Edit-In-Place mode.

```
leReturnToLevel(currWindowId)
```

leReturnToTop

```
leReturnToTop(
     [ w_windowId ]
     )
     => t / nil
```

Description

Applies to instances, nested instances, groups, and nested groups for which you are editing-in-place. When editing instances or nested instances in Edit-in-Place mode, returns to top level, exits Edit-in-Place mode. For groups, returns from the group, and if the group is nested, from all groups in the nesting, and exits Edit-in-Place mode. If you do not specify *w_windowId*, the current window is used.

Arguments

w_windowID	Database ID for the window in which to return during Edit-In-Place mode.

Value Returned

t	The return during Edit-in-Place mode was successful.
nil	The return during Edit-in-Place mode was not successful.

Example

```
leReturnToTop()
```

leSetStopLevelToEditLevel

```
leSetStopLevelToEditLevel (
     [ w_windowId ]
     )
     => x_level / nil
```

Description

Sets the display stop level to the current editing level.

Arguments

w_windowID	Window ID of the window specified.

Value Returned

x_level	Returns the level to which the display stop level has been set.
nil	The function is not successful.

Example

For the top level design, this function returns `0`. On EIP into a one-level design, this function returns `1`.

```
leSetStopLevelToEditLevel()
```

leSizeShape

```
leSizeShape(
     d_shapeId
     g_resize
     )
     => l_shapes / nil
```

Description

Resizes a shape by the specified amount. A shape can be resized uniformly in all directions or differently in different directions.

Arguments

d_shapeId	The database ID of the shape to be resized.
g_resize	Specifies the amount by which the shape is to be resized. The shape can be resized uniformly in all directions by specifying a single value. The shape can also be resized in specific directions by specifying a list of values in the following sequence: left, right, top, and bottom. For example, (1 2 0.5 1). Positive values increase the size of the shape; negative values decrease the size of the shape. Valid Values: any positive or negative number

Value Returned

l_shapes	A list of the resulting shapes if they are resized.
nil	The shapes are not resized.

Example

Increases the size of the `shape1` shape by 2 user units in all directions:

```
leSizeShape( shape1 2 )
```

Increases the size of the `shape1` shape by user units in the following directions: 2 on the left, 1 on the right, 3 at the top, and 1 at the bottom:

```
leSizeShape( shape1 '(2 1 3 1))
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the shape to size and specify resize value. If you do not specify *w_windowId*, the current window is used.

```
leHiSize( [ w_windowId ] ) => t / nil
```

leSlice

```
leSlice(
     l_points
     [ ?isHorizontal g_isHorizontal ]
     )
     => l_bBox /nil
```

Description

Performs the vertical slicing of a shape.

Arguments

l_points	List of points of any shape.
?isHorizontal *g_isHorizontal*	Performs the vertical or horizontal slicing of a shape. If this argument is not specified or is specified as `nil`, the function performs vertical slicing. If this argument is specified as `t`, the function performs horizontal slicing.

Value Returned

l_bBox	List of boxes formed from the shape after vertical slicing.
`nil`	The shapes was not sliced.

Example

```
leSlice(list('(0 0) '(20 0) '(20 20) '(10 20) '(10 10) '(0 10)))
```

The `leSlice` function returns the following values:

```
(((0 0) (10 10)) ((10 0) (20 20)))
```

leSplitShape

```
leSplitShape(
     d_shapeId
     l_delta
     l_splitLine
     g_leftOrRight
     [ g_lockAngles ]
     )
     => t / nil
```

Description

Splits a shape and moves one portion of it. Uses a multi-segment split line to divide the shape.

Arguments

d_shapeId	The database ID of the shape you want to redraw.
l_delta	List specifying distances in the X and Y direction to move the split portion.
l_splitLine	List of points defining the split line.
g_leftOrRight	Boolean specifying whether the part to be moved is to the left or right of the split line. Left and right are defined by the order of the points in the split line. Valid Values: `t` (left) or `nil` (right)
g_lockAngles	Boolean specifying whether angles between edges should be preserved. Default Value: `t` Valid Values: `t` or `nil`

Value Returned

`t`	The operation was successful.
`nil`	The operation was not successful.

Example

Moves the split portion of selected shape 10 units in the X direction and 12 units in the Y direction. The stretch group is to the left of the split line and the angles of the object do not change.

```
leSplitShape(car(geGetSelectedSet()) list(10 12) list(0:0 0:10) t t)
```

leStretchFig

```
leStretchFig(
     d_shapeId
     l_delta
     l_freePoints
     [ g_lockAngles ]
     )
     => t / nil
```

Description

Stretches shape *d_shapeId* by the delta *l_delta.* Donuts cannot be stretched.

Arguments

d_shapeId	The database ID of the figure.
l_delta	List of the X and Y distances to stretch the figure. You can specify the X and Y distances in list format, `list ( 0 2 )` or in coordinate format, `0:2`.
l_freePoints	Boolean list indicating whether each point in the figure is free to be moved or not. The list must match the number and order of points in the figure. If all of the vertices of the figure are marked as free, the entire figure is moved. (You can use `geGetSelSetFigPoint(d_shapeId)` to return the list of corresponding Boolean to the point list of figure *d_shapeId*.)
g_lockAngles	Specifies that angles of partially selected figures should not be modified when the figures are stretched. This option has no effect on instances or arrays. Valid Values: `t` or `nil`

Value Returned

`t`	The figure is stretched.
`nil`	The figure is not stretched.

Example

Stretches the right edge of figure `shapeA` 2 units in the Y direction. Two points of the figure are free to move and the angles of the figure cannot be changed.

```
leStretchFig( shapeA list( 0 2 ) list(nil t t nil) t )
```

You can also use `geGetSelSet()` and `geGetSelSetFigPoint()` to get the shape ID and Boolean list as shown here:

```
leStretchFig( car(geGetSelSet()) 0:2
geGetSelSetFigPoint( car(geGetSelSet()) ) t )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the shape to stretch, specify the distance to stretch, and specify free points. If you do not specify *w_windowId*, the current window is used.

```
leHiStretch( [ w_windowId ] ) => t / nil
```

leStretchShape

```
leStretchShape(
     d_shapeId
     l_delta
     l_freePoints
     [ g_lockAngles ]
     )
     => t / nil
```

Description

Stretches shape *d_shapeId* by the delta *l_delta.* Donuts cannot be stretched.

Arguments

d_shapeId	The database ID of the shape.
l_delta	List of the X and Y distances to stretch the shape. You can specify the X and Y distances in list format, `list ( 0 2 )` or in coordinate format, `0:2`.
l_freePoints	Boolean list indicating whether each point in the shape is free to be moved or not. The list must match the number and order of points in the shape. If all of the vertices of the shape are marked as free, the entire shape is moved. (You can use `geGetSelSetFigPoint(d_shapeId)` to return the list corresponding to the point list of shape *d_shapeId*.)
g_lockAngles	Specifies that angles of partially selected shapes should not be modified when the shapes are stretched. This option has no effect on instances or arrays. Valid Values: `t` or `nil`

Value Returned

`t`	The shape is stretched.
`nil`	The shape is not stretched.

Example

Stretches the right edge of shape `shapeA` 2 units in the Y direction. Two points of the shape are free to move and the angles of the shape cannot be changed.

```
leStretchShape( shapeA list( 0 2 ) list(nil t t nil) t )
```

You can also use `geGetSelSet()` and `geGetSelSetFigPoint()` to get the shape ID and Boolean list as shown here:

```
leStretchShape( car(geGetSelSet()) 0:2
geGetSelSetFigPoint( car(geGetSelSet()) ) t )
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to point to the shape to stretch, specify the distance to stretch, and specify free points. If you do not specify *w_windowId*, the current window is used.

```
leHiStretch( [ w_windowId ] ) => t / nil
```

leUnfreezeInst

```
leUnfreezeInst(
     d_instId
     )
     => t / nil
```

Description

Unfreezes or converts the non-Pcell instance, *d_instId*, to its original Pcell instance. When an instance is unfreezed the *leUnfreezeInst()* function only re-evaluates the Pcell instance but does not explicitly call the CDF callbacks.

This function is supported for mosaics.

Arguments

d_instId	The database ID of the instance to be converted to its original Pcell instance.

Value Returned

t	The non-Pcell instance is converted to its original Pcell instance.
nil	The non-Pcell instance is not converted to its original Pcell instance.

Example

Converts the non-Pcell instance, inst, to its original Pcell instance.

```
leUnfreezeInst(inst)
```

leUpdateInstanceCDFParameterValues

```
leUpdateInstanceCDFParameterValues(
     d_cellViewId
     d_instId
     l_paramNameList
     )
     => t / nil
```

Description

Updates the list of CDF parameters name-value pairs for the specified cellview and instance.

Arguments

d_cellViewId	The database ID of the cellview.
d_instId	An instance in the cellview.
l_paramNameList	A list CDF parameters and their values.

Value Returned

t	The CDF parameters are updated.
nil	The CDF parameters are not updated.

Examples

Returns `t` to indicate that the list of CDF parameters and values specified by `paramNameList` were updated in the `inst` instance of the `cv` cellview.

```
paramNameList = list(list("w" "240n) list("fw" "480n") list("fingers" "3"))
leUpdateInstanceCDFParameterValues(cv inst paramNameList)
```

leYankFigs

```
leYankFigs(
     d_cellViewId
     l_ptArray
     x_nLevels
     [ l_refPt ]
     [ x_nSides ]
     )
     => t / nil
```

Description

Copies specified objects into a yank buffer for later pasting. Copies objects in cellview *d_cellViewId* that fall within the shape defined by the points in *l_ptArray* and that are in the levels of hierarchy specified by *x_nLevels*. Optionally uses *l_refPt* as the reference point for the yank and *x_nSides* as the number of polygon sides to use when converting objects intersected by the yank shape.

For more information about placing the contents of the yank buffer into a cellview, see lePasteFigs.

Arguments

d_cellViewId	Database ID for the cellview containing the objects to yank.
l_ptArray	List of the coordinates defining the yank shape.
x_nLevels	Specifies the number of hierarchy levels to traverse.
l_refPt	A reference point used for later placing the yanked objects with `lePasteFigs`. Default Value: Coordinates of the lower left point of the bounding box for the yank shape Valid Values: coordinates of any point
x_nSides	Specifies the number of sides to use when converting objects to polygons. Default Value: `20` Valid Values: any positive integer

Value Returned

`t`	The objects are copied.
`nil`	The objects are not copied.

Examples

Copies all objects and partial objects in the current cellview that fall in the square region 0:0 to 10:10 into a buffer for future pasting. Objects and partial objects are copied down to two levels.

```
leYankFigs( geGetEditCellView() list( 0:0 10:0 10:10 0:10 ) 2 )
```

Interactive Function

Enter this function with the window ID argument only; the system prompts you to enter the coordinates of the yank shape. If you do not specify *w_windowId*, the current window is used.

```
leHiYank( [ w_windowId ] ) => t / nil
```

Selection Functions

Most of the selection functions are global functions used by all applications. Layer selectability is independent of the layer used for shape creation. Each library has its own entry layer, which is the layer that all shape creation functions use.

Related Topics

leApplyAreaFunction

leApplyLastCopyTransform

leFullSelectFigOfSelSet

leGetAreaEstimationDisplayNames

leApplyAreaFunction

```
leApplyAreaFunction(
     s_funcs_funcSymbol
     d_databaseId
     l_paramList
     )
     => f_result / nil
```

Description

Runs the area estimation function (provided as a function symbol) with provided parameters and database Id. This function is added as a part of Area Estimation consolidation.

Arguments

s_funcs_funcSymbol	Text string specifying functional callback symbol.
d_databaseId	Database Id.
l_paramList	List of parameters.

Value Returned

`t`	Returns `floating point number` if the function is successful.
`nil`	Returns `nil` if the function is not successful.

Example

Once you have the function callback symbol, the dbId, and the updated parameter list

```
if((funcRunMode == "direct"
```

then

//Collect all arguments and execute the function

```
area Value = leApplyAreaFunction(funcCallbackSymbol clusterId
funcParamListUpdated)
```

//Fill this in any form field

```
form->areaFD = areaValue
```

else

//Store funcCallBackSymbol, dbId and parameter list in internal data struct to be called later.

leApplyLastCopyTransform

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

Description

Copies the selected objects and applies the same transform as done in the last copy command and selects the newly created objects.

Note: If the last copy command created an array, then the `leApplyLastCopyTransform` function will apply the transformation of the previous copy command which did not create an array.

Arguments

None

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

```
leApplyLastCopyTransform()
```

leCLSCopyLayerShapes

```
leCLSCopyLayerShapes(
     t_cellName
     l_constraints
     [ l_deleteprevious ]
     )
     => l_shapes / nil
```

Description

Copies all shapes from one layer to another for which you have set appdef. This function provides an option to delete all shapes that were created on the specified ToLpps before creating new ones.

Arguments

t_cellName	Specifies the name of the cellview.
l_constraints	Specifies list of constraints on which the layers are to be copied.
l_deleteprevious	Delete all shapes and mask elements that were created on the specified ToLpps before creating new ones.

Value Returned

l_shapes	The list of shapes that are copied.
nil	Shapes have not been copied.

Examples

Returns a list of shapes that were copied from one layer to another.

```
leCLSCopyLayerShapes(gec()list(list("pg1" "METAL2 drawing"
"" list() list() "METAL3 drawing")) (( "METAL3 drawing" "mask1" ))
=> (db0x9090x)
```

leFullSelectFigOfSelSet

```
leFullSelectFigOfSelSet(
     [ w_windowId ]
     )
     => t
```

Description

Fully selects all the partially selected figures. If w_windowId is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command was successful.

leGetAreaEstimationDisplayNames

```
leGetAreaEstimationDisplayNames(
     t_aliasName
     l_legalObjectTypes
     )
     => t_funcSymbol / nil
```

Description

Retrieves area estimation function alias name list defined for the specified object types. The object types supported are 'cellView, 'inst, cluster, 'group. This function is added as a part of Area Estimation consolidation.

Arguments

t_aliasName	Text string specifying a simple display name for a cyclic list in the graphical user interface.
l_legalObjectTypes	List of legal object types for which this function will work. Valid Values: 'cellview, 'cluster, 'group, 'inst

Value Returned

t_funcSymbol	Returns *t_funcSymbol* if the function is successful.
nil	Returns nil if the function is not successful.

Example

```
funcCallbackSymbol = leGetAreaEstimatorFunction( ?aliasName "simpleAreaEst"
?legalObjectTypes)
```

Once you have selected a particular area estimator display or alias name, the invocation code or *Apply* button callback returns functional symbol as in the example => 'myFunction

leGetAreaEstimatorFunction

```
leGetAreaEstimatorFunction(
     t_AliasName
     l_legalObjectTypes
     )
     => t_funcSymbol / nil
```

Description

Retrieves the function symbol of the area estimation function registered for alias name and object type. This function is added for Area Estimation Consolidation.

Arguments

t_AliasName	Text string specifying a simple display name for a cyclic list in the graphical user interface.
l_legalObjectTypes	List of legal object types for which this function will work. Valid Values: ‘cellview, ‘cluster, ‘group, ‘inst

Value Returned

t_funcSymbol	Returns *t_funcSymbol* if the function is successful.
nil	Returns nil if the function is not successful.

Example

```
funcCallbackSymbol = leGetAreaEstimatorFunction( ?aliasName "simpleAreaEst"
?legalObjectTypes)
```

Once you have selected a particular area estimator display or alias name, the invocation code or *Apply* button callback returns functional symbol as in the example => ‘myFunction’.

leGetAreaEstimatorParamList

```
leGetAreaEstimatorParamList(
     t_functionAliasName
     l_legalObjectTypes
     )
     => l_funcParamList / nil
```

Description

Retrieves the parameter list of the area estimation function registered for alias name and object type. This function is added as a part of area estimation consolidation.

Arguments

t_functionAliasName	Text string specifying a simple display name for a cyclic list in the graphical user interface.
l_legalObjectTypes	List of legal object types for which this function will work. Valid Values: ‘cellview, ‘cluster, ‘group, ‘inst

Value Returned

l_funcParamList	Returns *l_funcParamList* if the function is successful.
nil	Returns nil if the function is not successful.

Example

Gets the parameter list if present.

```
funcParam List = leGetAreaEstimatorParamList( ?aliasName" "simpleAreaEst"
?legalObjectTypes '("cluster") )
=> '((pinAreaFactor 10.0) (allowPinArea nil))
```

leGetAreaEstimatorRunMode

```
leGetAreaEstimatorRunMode(
     t_aliasName
     l_legalObjectTypes
     )
     => t_RunMode / nil
```

Description

Retrieve the Run Mode list of the area estimation function registered for alias name and object types. This function is added as a part of area estimation consolidation.

Arguments

t_aliasName	Text string specifying a simple display name for a cyclic list in the graphical user interface.
l_legalObjectTypes	List of legal object types for which this function will work. Valid Values: ‘cellview, ‘cluster, ‘group, ‘inst

Value Returned

t_RunMode	Returns *t_RunMode* if the function is successful.
nil	Returns nil if the function is not successful.

leGetEditFigGroup

```
leGetEditFigGroup(
     [ w_windowID ]
     )
     => d_figGroupId / nil
```

Description

Returns the database ID of the currently active figure group for Edit-In-Group mode editing, in the specified window or the current window.

Arguments

w_windowID	ID of the window containing the desired figure group. If you do not specify *w_windowId*, the current window is used.

Value Returned

d_figGroupId	Database ID of the currently active figure group for Edit-In-Group mode editing.
nil	The function failed.

Example

For the specified window ID, returns database ID of the currently active figure group for Edit-In-Group mode editing.

```
leGetEditFigGroup(currWindowId)
```

leGetEntryLayer

```
leGetEntryLayer(
     [ d_techFileId ]
     )
     => l_layerPurposePair / nil
```

Description

Returns the entry layer for technology file *d_techFileId* as a list containing the layer name and layer purpose. If *d_techFileId* is not specified, the current technology file is used.

Arguments

d_techFileId	Database ID for the technology file specified.

Value Returned

l_layerPurposePair	A list of the layer name and purpose if an entry layer for technology file *d_techFileId* exists.
nil	The technology file *d_techFileId* does not exist.

Example

Returns the layer name and layer purpose for the entry layer in the technology file myTech.

```
leGetEntryLayer( myTech )
```

leGetObjectSelectable

```
leGetObjectSelectable(
     t_objectName
     )
     => t / nil
```

Description

Checks whether or not objects are set to selectable in the LSW.

Arguments

t_objectName	Text string specifying the object you want to check the selectability of. Valid Values: "inst", "marker", "pin" (pin object), "via", "Groups", "fluidShape", "row", "P&R", "Snap", "Area", "Cluster", "SoftBlock P&R", "SoftBlock Snap", "SoftBlock Pin", "Halo", "Placement", "Routing", "Fill", "Slot", "Pin" (pin blockage object), "Feedthru", "Screen", "tracks", "RoutingGrid", "PlacementGrid", "ConAxes", "Circle/Ellipse", "Donut", "Label", "Path", "PathSeg", "Polygon", "Rectangle", "Mosaic", Markers ("Acknowledge Warning", "Annotation", "Critical Error", "Error", "Fatal", "Info", "Signed Off Critical Error", "Signed Off Error", "Warning").

Value Returned

t	Returns t if the object is set to selectable in the LSW.
nil	Returns nil if the object is not set to selectable in the LSW.

Example

Returns t if the P&R Boundary object is set to selectable in the LSW and nil if it is not.

```
leGetObjectSelectable("P&R")
```

leGetObjectVisible

```
leGetObjectVisible(
     t_objectName
     )
     => t / nil
```

Description

Checks whether or not objects are set to visible in the LSW.

Arguments

t_objectName	Text string specifying the object you want to check the visibility of. Valid Values: "inst", "marker", "pin" (where "pin" is pin object), "track", "row", "via", "P&R", "Snap", "Cluster", "Area", "Placement", "Routing", "Wiring", "Slot", "Fill", "Feedthru", "Screen", "Pin" (where "Pin" is a pin blockage object.), "Mixed Row", "Custom Placement Area", "Placement Grid", "Routing Grid", "Halo", "Groups", "ConAxes", "Logical","Physical", "Steiner", "Snap Patterns".

Value Returned

t	Returns t if the object is set to visible in the LSW.
nil	Returns nil if the object is not set to visible in the LSW.

Example

Returns t if the P&R Boundary object is set to visible in the LSW and nil if it is not.

```
leGetObjectVisible("P&R")
```

leGetSnapOptions

```
leGetSnapOptions
     [ w_windowID ]
     )
     => l_dpl / nil
```

Description

Generates the default snap options in the form of DPLs (Disembodied Property Lists). If a valid window ID is specified in the input list and it is a layout window, the options are listed considering window variables. If a window ID is not specified, the default values of environment variables are listed.

Arguments

w_windowId	Database ID of the window.

Value Returned

l_dpl	The list of DPLs of the default snap options.
nil	The command is not successful.

Example

```
win = hiGetCurrentWindow()
snapOptionWin = leGetSnapOptions(win)
(nil toSP t toWSP t toRow nil toPlacementGrid nil toMfgGrid nil toXYGrid nil
wspDetail t
wspDetailSnapAllShapes nil snapMethod "defaultDetailOverride" snapGrid "Any"
snapDirection "nearest" snapXEdge "low" snapYEdge "low" snapDepth 31 freeSnapDepth
31
fixedSnapDepth 31 matchPinColor nil xSnapSpacing 0.001 ySnapSpacing 0.001
fastDetailModeSnapping t abstractModeSnapAnyBoundaryEdge nil allSPGridsEnclosures
t
snapToXYGridFirst nil boundarySnapXGrids "" boundarySnapYGrids ""
boundarySnappingLayer "" ignoreParallelMinWidthForSquarePinResize nil
rowSiteSymmetry nil snapDeviceInDetailMode t snapStdCellInDetailMode nil
ignoreRef t propBasedInstSnap t polyGridFeatures t)
```

leGetSnapTransform

```
leGetSnapTransform
     d_figToSnap
     d_targetRow | nil
     l_projectionTransform
     l_ignoreFigs
     l_options
     g_detailedResults
     )
     => l_result / nil
```

Description

Returns snap transform for the figure. The cellview of the figure is the one where the snapping is taking place. This function does not modify figure or cellview so it works for read only cellview as well. The cellview and the specified options are used to determine the licenses needed for the function to run. If the licenses can be checked out, the function runs. If not, the return value signifies failure because of licensing.

Arguments

d_figToSnap	The database figure to be snapped. Snapping is done in the cellview of the figure.
d_targetRow|nil	The instance snaps to the target row if specified else it searches for a valid row. The target row can be dbRow, dbPlaceArea, a row in a row region, and an area bounday figure which is inside the dbPlaceArea or an oaRow inside a dbRow. If the target row is a random dbId, the function returns nil.
l_projectionTransform	A list that specifies the projection transform which when concatenated to the database position of the figure to snap is the new position where snapping is performed.
l_ignoreFigs	A list that specifies the set of figures to be ignored for snapping.

l_options

This is a DPL (disembodied property list) with options to set snap input parameters.

DPL list should look like:

```
l_options = '( nil
toSP t|nil
toWSP t|nil
toRow t|nil
toPlacementGrid t | nil
toMfgGrid t | nil
toXYGrid t | nil
toWSPDetail t | nil
toWSPDetailSnapAllShapes t | nil
snapMethod  "defaultAbstractOverride" |
"defaultDetailOverride" |"abstractOnly" |
"detailOnly"
snapGrid "any" | "local" | "global"
snapDirection "nearest" | "low" | "high"
snapXEdge  "low" | "high"
snapYEdge  "low" | "high"
snapDepth  0 | 0-31
freeSnapDepth 0 | 0-31
fixedSnapDepth 0 | 0-31
matchPinColor t | nil
xSnapSpacing  0.001 | any value multiple of
mfg grid
ySnapSpacing  0.001 | any value multiple of
mfg grid
fastDetailModeSnapping  t | nil
abstractModeSnapAnyBoundaryEdge t | nil
allSPGridsEnclosures t | nil
snapToXYGridFirst t | nil
boundarySnapXGrids  "" | t_names
boundarySnapYGrids  "" | t_names
boundarySnappingLayer "" | t_names
ignoreParallelMinWidthForSquarePinResize t |
nil
rowSiteSymmetry t | nil
snapDeviceInDetailMode t | nil
ignoreRef t | nil
propBasedInstSnap t | nil
polyGridFeatures t | nil
)
```

g_detailedResults	This boolean flag specifies if detailed snap data is displayed in the result.

Value Returned

l_result	Displays the result as: `list (nil) | list (transform | detailed snap result + transform)`
`nil`	The command is not successful.

Examples

```
xform = leGetSnapTransform(inst nil list(list(0 0) "R0") list(nil) options t)
    ======================= START SNAP INFO =======================
    Snap Mode = Detail
    Axis snapped-to = Horizontal
    SPDefName snapped to = GPG90 (PPitch, poly90)
    Layer of snapped-to LPP = Poly
    Purpose of snapped-to LPP = drawing
    Enclosure = 0.009000
    Bounding box for snapped-to LPP=((34.791000 13.957000) (35.349000 14.211000))
    ======================= END SNAP INFO =======================
    ((-0.015 0.0) "R0" 1.0)
```

Error due to licensing

```
    nil
    A valid error message - "Valid license needed to run leGetSnapTransform API."
```

Non-detailed result

Successful

```
   xform = leGetSnapTransform(inst nil list(list(0 0) "R0") list(nil) options nil)
    ((-0.015 0.0) "R0" 1.0)
```

Not successful

```
   xform = leGetSnapTransform(inst fakeRow list(list(0 0) "R0") list(nil) options
nil)
    nil
```

leGetTechFormList

```
leGetTechFormList(
     d_techFileId
     )
     => l_formIds / nil
```

Description

Returns a disembodied property list (DPL) that stores the dynamic form Ids associated with a technology library opened in a Virtuoso session. If the forms have not been opened yet, the SKILL function creates the forms and returns the form ID list.

Arguments

d_techFileId	Database ID of the technology library

Value Returned

l_formIds	DPL of form Ids associated with the specified technology library.
nil	DPL of form Ids is not returned.

Example

Returns the DPL of dynamic form Ids associated with the `techId` technology library.

```
formList=leGetTechFormList(techId)
```

You can then access the forms, say the *Move* form, in the following manner:

```
formList->moveForm
```

leGetValidLayerList

```
leGetValidLayerList(
     [ d_techFileId ]
     )
     => l_layerIdList / nil
```

Description

Returns a list of the valid entry layers for technology file *d_techFileId*. A layout view must be open for `leGetValidLayerList` to return the list of valid entry layers.

Arguments

d_techFileId	Database ID for the technology file specified.

Value Returned

l_layerIdList	A list of the valid entry layers for technology file *d_techFileId*.
nil	The technology file *d_techFileId* does not exist, or the entry layers are not defined.

Example

```
leGetValidLayerList( myTech )
```

Returns the following list of layer-purpose pairs:

```
(nwell drawing
 pwell drawing
 ndiff drawing
 pdiff drawing
 diff drawing
 nsd drawing
 psd drawing
 Ptap drawing
 Ntap drawing
 nimplant drawing
 pimplant drawing
 poly1 drawing
 poly2 drawing
 metal1 drawing
 metal2 drawing
 metal3 drawing
 via drawing
 via2 drawing)
```

leGetValidPurposeList

```
leGetValidPurposeList(
     d_techFileId
     )
     => l_purposeIdList / nil
```

Description

Returns a list of all valid purposes for the specified technology file.

Arguments

d_techFileId	ID of technology file

Values Returned

l_purposeIdList	A list of valid purposes is returned when the command executes successfully
nil	Returned when the command fails to run

Example

```
techId = techGetTechFile(geGetEditCellView())
leGetValidPurposeList(techId)
```

Outputs a list of purposes in the technology file returned by techGetTechFile.

```
("drawing" "fill" "slot" "OPCSerif" "OPCAntiSerif"
    "annotation" "gapFill" "redundant" "fillOPC" "blockage"
    "fatal" "critical" "soCritical" "soError" "ackWarn"
    "info" "track" "grid" "warning" "tool1"
    "tool0" "label" "flight" "error" "annotate"
    "drawing1" "drawing2" "drawing3" "drawing4" "drawing5"
    "drawing6" "drawing7" "drawing8" "drawing9" "boundary"
    "pin" "net" "cell"
)
```

leIsFigSelectable

```
leIsFigSelectable(
     d_figId
     )
     => t / nil
```

Description

Checks whether the figure *d_figId* is selectable.

Arguments

d_figId	The database ID of the figure to check.

Value Returned

t	The figure *d_figId* is selectable.
nil	The figure *d_figId* is not selectable.

Example

Returns t if figure fig4 is selectable.

```
leIsFigSelectable( fig4 )
```

leIsInstSelectable

```
leIsInstSelectable(
     [ d_techFileId ]
     )
     => t / nil
```

Description

Checks whether the instances in *d_techFileId* are selectable. If *d_techFileId* is not specified, the current technology file is used.

Arguments

d_techFileId	The database ID of the technology file to check.

Value Returned

t	The instances in *d_techFileId* are selectable.
nil	The instances in *d_techFileId* are not selectable.

Example

Returns t if instances in myTech are selectable.

```
leIsInstSelectable( myTech)
```

leIsLayerSelectable

```
leIsLayerSelectable(
     l_layerPurposePair
     [ d_techFileId ]
     )
     => t / nil
```

Description

Checks whether the layer *l_layerPurposePair* is selectable.

Arguments

l_layerPurposePair	Layer-purpose pair to check for selectability.
d_techFileId	The database ID of the technology file containing the layer-purpose pair.

Value Returned

t	The layer *l_layerPurposePair* is selectable.
nil	The layer *l_layerPurposePair* is not selectable.

Example

Returns `t` if layer-purpose pair `poly1 drawing` is selectable.

```
leIsLayerSelectable( list("poly1" "drawing") )
```

leIsLayerValid

```
leIsLayerValid(
     l_layerPurposePair
     [ d_techFileId ]
     )
     => t / nil
```

Description

Checks whether layer *l_layerPurposePair* is a valid layer. If *d_techFileId* is not specified, the current technology file is used.

Arguments

l_layerPurposePair	Layer-purpose pair to be checked for validity.
d_techFileId	The database ID of the technology file that is checked.

Value Returned

`t`	The layer *d_layerPurposePair* is a valid layer.
`nil`	The layer *d_layerPurposePair* is not a valid layer.

Example

Returns `t` if layer-purpose pair `metal1 drawing` in `lib2` is a valid layer.

```
leIsLayerValid( list("metal1" "drawing") lib2 )
```

leIsLayerVisible

```
leIsLayerVisible(
     l_layerPurposePair
     [ d_techFileId ]
     )
     => t / nil
```

Description

Checks whether the layer specified by *l_layerPurposePair* is visible. If *d_techFileId* is not specified, the current technology file is used.

Arguments

l_layerPurposePair	Layer-purpose pair checked for visibility.
d_techFileId	The database ID of the technology file containing the layer-purpose pair checked for visibility.

Value Returned

t	The layer *l_layerPurposePair* is visible.
nil	The layer *l_layerPurposePair* is not visible.

Example

Returns t if layer1 is a visible layer.

```
leIsLayerVisible( list("layer1" "drawing") )
```

leIsNewODCInfraEnabled

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

Description

Checks if the new `odcInfra` is enabled or not in the current Virtuoso session.

Arguments

None

Value Returned

`t`	The new `odcInfra` is enabled in the current Virtuoso session.
`nil`	The new `odcInfra` is not enabled in the current Virtuoso session. The old `odcInfra` is enables.

Example

```
leIsNewODCInfraEnabled()
```

Returns `t`.

leIsPinSelectable

```
leIsPinSelectable(
     [ d_techFileId ]
     )
     => t / nil
```

Description

Checks whether the pins in *d_techFileId* are selectable. If *d_techFileId* is not specified, the current technology file is used.

Arguments

d_techFileId	The database ID of the technology file to check.

Value Returned

t	The pins in library *d_techFileId* are selectable.
nil	The pins in library *d_techFileId* are not selectable.

Example

Returns t if pins in myTechFile are selectable.

```
leIsPinSelectable( myTechFile )
```

leIsPointInsideFig

```
leIsPointInsideFig(
     d_figId
     l_point
     )
     => t / nil
```

Description

Checks if the specified point is inside the figure.

Arguments

d_figId	Database ID of the figure
l_point	Coordinates of the input point

Value Returned

t	Returned if the point *l_point* lies inside the figure.
nil	Returned if the point *l_point* does not lie inside the figure.

Example

Checks if the point *x*=5, *y*=5 is inside the first figure of the selection set.

```
leIsPointInsideFig(car(geGetSelSet()) list(5 5))
```

leLSWGetBlockageSelectable

```
leLSWGetBlockageSelectable(
     t_blockageTypeName
     t_layerName
     )
     => t / nil
```

Description

Indicates whether a layer blockage is selectable.

Arguments

t_blockageTypeName	
	Any valid layer blockage type. Blockages can have the following types: "Routing", "Wiring", "Fill", "Slot", "Pin", "Feed", and "Screen".
t_layerName	Any valid layer name. Example: metal1.

Value Returned

`t`	The blockage is selectable.
`nil`	The blockage is not selectable.

Example

Returns `t` if "metal1" routing blockage is selectable and `nil` if it is not.

```
leLSWGetBlockageSelectable("Routing" "metal1")
```

leLSWGetBlockageVisible

```
leLSWGetBlockageVisible(
     t_blockageTypeName
     t_layerName
     )
     => t / nil
```

Description

Sets whether a layer blockage is visible or not. If the blockage is successfully set to visible the function returns t. If it is not set to visible the function returns nil.

Arguments

t_blockageTypeName	Any valid layer blockage type. Blockages can have the following types: "Routing", "Wiring", "Fill", "Slot", "Pin", "Feed", and "Screen".
t_layerName	Any valid layer name. Example: metal1.

Value Returned

`t`	The blockage is visible.
`nil`	The blockage is not visible.

Example

Returns `t` if "metal1" routing blockage is visible and `nil` if it is not.

```
leLSWGetBlockageVisible("Routing" "metal1")
```

leLSWGetRoutingGridVisible

```
leLSWGetRoutingGridVisible(
     t_layerName
     )
     => t / nil
```

Description

Returns the visibility of the specified routing grid layer. For the layer specified, the following must be defined in the technology file: in the `layerRules` section, for `routingDirections`, the direction for the layer; in the `constraintGroups` section, for `routingGrids`, values for the `horizontalPitch` and `verticalPitch`. In the LSW, you can check this visibility by clicking on the *Grid* button, then on the *Routing Grid* button.

Arguments

t_layerName	A text string specifying any valid layer name. Example: "`metal1`"

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

Returns `t` if "metal1" routing grid is visible and `nil` if it is not.

```
leLSWGetRoutingGridVisible("metal1")
```

leLSWSetBlockageSelectable

```
leLSWSetBlockageSelectable(
     t_blockageTypeName
     t_layerName
     g_status
     )
     => t / nil
```

Description

Sets whether a layer blockage is selectable or not. If the blockage is successfully set to selectable the function returns t. If it is not set to selectable the function returns nil.

Arguments

t_blockageTypeName	Any valid layer blockage type. Blockages can have the following types: "Routing", "Wiring", "Fill", "Slot", "Pin", "Feedthru", and "Screen".
t_layerName	Any valid layer name. Example: metal1.
g_status	Status is `t` or `nil`.

Value Returned

t	The blockage is selectable.
`nil`	The blockage is not selectable.

Example

Returns `t` if "metal1" routing blockage is successfully set to selectable and `nil` if it is not.

```
leLSWSetBlockageSelectable("Routing" "metal1" t)
```

leLSWSetBlockageVisible

```
leLSWSetBlockageVisible(
     t_blockageTypeName
     t_layerName
     g_status
     )
     => t / nil
```

Description

Sets whether a layer blockage is visible or not. If the blockage is successfully set to visible the function returns t. If it is not set to visible the function returns nil.

Arguments

t_blockageTypeName	Any valid layer blockage type. Blockages can have the following types: "Routing", "Wiring", "Fill", "Slot", "Pin", "Feedthru", and "Screen".
t_layerName	Any valid layer name. Example: metal1.
g_status	Status is t or nil.

Value Returned

t	The blockage is visible.
nil	The blockage is not visible.

Example

Returns t if "metal1" routing blockage is successfully set to visible and nil if it is not.

```
leLSWSetBlockageVisible("Routing" "metal1" t)
```

leLSWSetRoutingGridVisible

```
leLSWSetRoutingGridVisible(
     t_layerName
     g_status
     )
     => t / nil
```

Description

Sets whether a specific routing layer grid is visible. For the layer specified, the following must be defined in the technology file: for `routingDirections` in the `layerRules` section, the direction for the layer; for `routingGrids` in the `constraintGroups` section, values for the `horizontalPitch` and `verticalPitch`. In the LSW, you set this visibility by clicking on the *Grid* button.

Arguments

t_layerName	A text string specifying any valid layer name. Example: "`metal1`"
g_status	Boolean specifying whether the routing grid is visible. Valid Values: `t` for visible or `nil` for invisible

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

Returns `t` if "metal1" routing grid is successfully set to visible and `nil` if it is not.

```
leLSWSetRoutingGridVisible("metal1" t)
```

leLSWSetTrackPatternVisible

```
leLSWSetTrackPatternVisible(
     g_status
     )
     => t / nil
```

Description

Sets the status of the visibility of track patterns and the *Track* grid visibility button in the LSW. This function does not affect the visibility of individual track layer-purpose pairs; use the leSetObjectVisible function to control LLP visibility. To turn track grid visibility on or off using the graphical user interface, in the LSW window, click *Grid*, then check the *Vis* box next to *Track*.

Arguments

g_status	Whether to turn on (`t`), or off (`nil`) visibility. Valid Values: `t` or `nil`

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

Sets track patterns to visible; in the LSW, under *Grid*, sets the *Track* object button to visible.

```
leLSWSetTrackPatternVisible ( t )
```

To contrast changing the visibility of track patterns versus the visibility of LPPs, assume you have "metal1" track patterns and a rectangle on "metal1/track". Calling `leSetObjectVisible("tracks" nil)` makes both track patterns and the rectangle invisible, while calling `leLSWSetTrackPatternVisible(nil)` makes only the track patterns invisible, leaving the rectangle visible.

lePrintHierarchyTree

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

Description

Opens a form to choose library, cell, and view and then generates the hierarchy information.

Arguments

None

Value Returned

`t`	Returns `t` if the function was successful.
`nil`	Returns `nil` if the function was not successful.

Example

```
lePrintHierarchyTree()
```

leRaiseLSW

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

Description

Raises the Layer Selection Window (LSW) to the top of the window stack. It can be used if the LSW has been iconified. If the LSW is already at the top of the window stack, this function has no effect.

Note: This SKILL function is not supported in Palette assistants. On execution of this SKILL function, the following warning message is displayed.
WARNING (LE-105566): The SKILL function leRaiseLSW cannot be used because the LSW is disabled. The Palette assistants are enabled as replacements for the LSW. To enable LSW, set the UNIX shell environment variable CDS_PALETTE_OFF before launching Virtuoso.

Arguments

None

Value Returned

`t`	The LSW is raised to the top of the window stack.
`nil`	The LSW is not raised to the top of the window stack.

leRegAreaEstimator

```
leRegAreaEstimator(
     s_functionSymbol
     t_functionAlias
     l_legalObjectTypes
     [ l_parameterNameValuePairs ]
     [ t_runMode ]
     )
     => t / nil
```

Description

Registers a particular area estimation function. This function is part of area estimation consolidation. The name of the estimator that will appear in the cyclic *Area Estimator* field in the Create Cluster Boundary form, and the function the software will invoke when you click the *Estimate* button on the form. While searching for available area estimators, the applications/UI code should know the requirements for object types; the application provides database IDs and executes the area estimator in direct or deferred mode.

Arguments

s_functionSymbol	User-defined function name.
t_functionAlias	Text string specifying a simple display name for a cyclic list in the graphical user interface.
l_legalObjectTypes	List of legal object types for which this function will work. The objects listed act as search keys for commands to filter out the list of relevant estimators. Valid Values: `'cellview, 'cluster, 'group, 'inst`
l_parameterNameValuePairs	(Optional) List of name-value pairs, where name specifies parameter name and value is the default value. Only simple the data types int, float, text, and boolean are supported. Parameters might be required internally by the area estimator for area calculation formulae.
t_runMode	Text string specifying how to call the area estimator. When run mode is `"direct"`, all information required for the area estimator is present; it can be run and the area value can be viewed. When the run mode is `"deferred"`, the user wants to set the area estimator and apply the parameters; all this information is stored and called later when all the information is available. If no run mode is specified, this function can be run both in direct and deferred mode. For example, in direct mode, when the input is a Schematic cellview ID, the area estimator traverses the entire hierarchy to output an area. In deferred mode, when the input is a Physical Layout cellview ID, the area calculation is deferred until a later step, such as the netlisting step of GFS. Once the physical hierarchy is present, area estimation can be called. Valid Values: "`direct`" or "`deferred`"

Value Returned

`t`	Returns `t` if the function was successful.
`nil`	Returns `nil` if the function was not successful.

Example 1

```
leRegAreaEstimator('myFunction
                   "simpleAreaEst"
                   '('cluster)
                   '((pinAreaFactor 10.0) (allowPinArea nil))
                   "direct"
                   )
```

The `simpleAreaEst` alias is registered for the `cluster` object type, for the area estimator function, which must be called in direct mode, with the following signature:

```
myFunction(ID @optional (pinAreaFactor 10.0) (allowPinArea t))
```

Example 2

```
leRegAreaEstimator('myComplexFunction
                   "complexAreaEst"
                   '('cellView 'cluster)
                   '((contactAreaFactor 1.0) (param t))
                   "deferred"
                   )
```

The `complexAreaEst` alias is registered for `cellview` and `cluster` object types for the area estimator function, which must be called in deferred mode, with the following signature:

```
myComplexFunction(ID @optional (contactAreaFactor 1.0) (param t)
```

leRegClusterBdyEstimator

```
leRegClusterBdyEstimator(
     t_displayName
     t_functionName
     )
     => t / nil
```

Description

Registers the name of the estimator which will appear in the cyclic field of the Area Estimator field in the Create Cluster Boundary form in the Floorplanning tool, and the function the software will invoke when you click the *Estimate* button on the form.

Arguments

t_displayName	Name that appears in the *Area Estimator* field in the Create Cluster Boundary form in Layout XL.
t_functionName	Name of the function that is run when you click *Estimate* on the form.

Value Returned

`t`	Returns `t` if the area estimator is successfully registered.
`nil`	Returns `nil` if the area estimator is not successfully registered.

Example

Registers `estimateFunc` as the name which appears in the cyclic field of the Area Estimator field in the Create Cluster Boundary form in the Floorplanning tool, and calls the function name the software will invoke when you click the Estimate button on the form (in this example, `myEstimatorFunc`).

```
leRegClusterBdyEstimator("estimateFunc" "myEstimatorFunc")
```

leRegisterPPNetNameFn

```
leRegisterPPNetNameFn(
     S_ppNetNameFn
     )
     => t / nil
```

Description

Registers a SKILL function that specifies the names of pseudo-parallel subnets when called by the layout generation commands *Generate All From Source* and *Update Components and Nets*.

This is useful when you work with the layout using non-Cadence tools or scripts that follow a set naming convention. Subnets specified by this SKILL function are created by dbCreateNamedSubNet and can be found using dbFindNetByName.

Arguments

S_ppNetNameFn	Name of the SKILL function to be registered. The SKILL function must take the following three arguments and return a string that is a valid net name that is not the name of an existing net or subnet in the same cellview. ■ `netId`: ID of the net for which a new pseudo parallel subnet is to be returned. ■ `num`: The unique number for each subnet that is created for the top level net. If the *Generate All From Source* command creates 4 subnets for the top level net `netA`, the registered *S_ppNetNameFn* is called 4 times during *Generate All From Source* for `netA`, and `num` has a different value between 1 and 4 for each call. ■ `isVector`: Specifies whether the top level `netId` is a vector bit net. Valid values: `t` or `nil`. This can be used to ensure a legal name is returned. For example, if the top level net is named `n<1>` then it is a vector bit net and `isVector` is `t`. The registered function ensures that it returns a valid name such as `na<1>` rather than `n<1>a` which is an illegal net name. If leRegisterPPNetNameFn is called with an empty string argument, then any existing named pseudo parallel net name function is unregistered and the pseudo parallel subnets are named automatically without callbacks to any user function.

Value Returned

t	The SKILL function was registered.
`nil`	The SKILL function registration failed.

Example

Registers `ppNetName` as the user-defined SKILL function.

```
leRegisterPPNetNameFn("ppNetName")
```

Where `ppNetName` is the function procedure defined as:

```
procedure(ppNetName(netId num isVector)
    let((name suffix parsed)
        if( num<=26 then
        suffix = intToChar(charToInt('a) + num-1)
        else
        suffix = sprintf(nil "_%d" num)
        )
        if( isVector then
        parsed = parseString(netId~>name "<>")
        name = strcat(car(parsed) suffix "(" cadr(parsed) ")")
        else
        name =  strcat(netId~>name suffix)
        )
    )

leRegisterPPNetNameFn("ppNetName")
```

leRegisterUseGravity

```
leRegisterUseGravity(
     t_cmdName
     )
     => t
```

Description

Enables a user-defined SKILL enter function to use the gravity feature. Gravity follows the options specified in the Layout Editor Options form.

Arguments

t_cmdName	The name of the command for which you want to use gravity when the user enters points.

Value Returned

t	Gravity is used as per the options specified in the Layout Editor Options form while the SKILL enter function runs.

Example

For the following user-defined SKILL enter function:

```
(defun MyStretchSpecial (...)
  ...
  (enterPoint ?cmdName "StretchSpecial" ...)
  ) ; defun
```

If you want to use the gravity feature for the function `StretchSpecial` when the command runs, use the following function:

```
(leRegisterUseGravity "StretchSpecial") ; Gravity now works for StretchSpecial
(MyStretchSpecial ...)
```

For as long as `MyStretchSpecial` function runs, the entered points will use gravity as per the options specified in the Layout Editor Options form.

leRegUserLayerSelectionFilter

```
leRegUserLayerSelectionFilter(
     t_filterName
     )
     => t / nil
```

Description

Registers a user-defined layer selection filter to prevent you from selecting shapes on the layers.

Arguments

t_filterName	Name of user defined SKILL function which takes LPP ID as argument.

Value Returned

`t`	Returns `t` if the filter is successfully registered.
`nil`	Returns `nil` if the filter is not successfully registered.

Example

Registers `layerFilter` as the user-defined SKILL function.

```
leRegUserLayerSelectionFilter("layerFilter")
```

Where `layerFilter` is the function procedure defined as:

```
procedure(layerFilter(lpId)
    let(()
       cond(
           (lpId~>name == "metal1"
           nil
           )
           (t
           t
           )
        )
    )
```

leRegUserObjectSelectionFilter

```
leRegUserObjectSelectionFilter(
     t_filterName
     )
     => t / nil
```

Description

Registers a user defined object selection filter to prevent you from selecting objects individually or as part of a selected set.

Arguments

t_filterName	Name of user defined SKILL function which takes figure ID as argument, returns `t` if the given object is selectable or `nil` if not selectable.

Value Returned

`t`	Returns `t` if the filter is successfully registered.
`nil`	Returns `nil` if the filter is not successfully registered.

Example

Registers "`objectFilter`" as the user defined SKILL function.

```
leRegUserObjectSelectionFilter("objectFilter")
```

Where `objectFilter` is the function procedure defined as:

```
procedure(objectFilter(figId)
    let(()
       cond(
           (figId~>name == "I1"
           nil
           )
           (figId~>objType == "rect"
           nil
           )
           (t
           t
```

)
)
)

leRepeatCopyMoveStretch

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

Description

Copies, moves, or stretches the selected objects and applies the same transform as done in the last copy, move, or stretch command and selects the newly created objects. This function applies to all the options specified in the last copy, move, or stretch commands, including the *Create Synchronous Copy* option available only in Layout XL and higher tiers.

Note: The function does not support the *Change To Layer* option of the copy and move commands.

Arguments

None

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

If there are two rectangles on the canvas and you start the `Copy` command and copy the rectangles. Then, you invoke the `leRepeatCopyMoveStretch ()` function in the CIW, the selected rectangles are copied again with the last transformation. Similarly, if you move the rectangles and invoke the `leRepeatCopyMoveStretch ()` function, the two selected rectangles move with the same last transformation. The behavior is similar for the `Stretch` command.

```
leRepeatCopyMoveStretch()
```

leReportTrimmedShapesInCustomStyle

```
leReportTrimmedShapesInCustomStyle(
     d_cvId
     [ ?layerName t_layerName ]
     [ ?depth x_depth ]
     [ ?createMarker g_createMarker ]
     [ ?printReport g_printReport ]
     [ ?showReportWindow g_showReportWindow ]
     [ ?saveReport t_saveReport ]
     [ ?markerSeverity t_markerSeverity ]
     )
     => l_lists / nil
```

Description

Reports pathsegs on the specified layer that have overlapping trim shapes and pathsegs that are abutted to trim shapes but do not have a relevant patch type. The function also reports the following: polygons, rectangles, and paths abutting or overlapping trim shapes, trim shapes that are abutted to pathsegs but do not have a bridge metal, and the trim shapes that are not abutted to pathsegs, polygons, rectangles, or paths in the same cellview. The function searches for trim shapes and metal shapes in the hierarchy up to the depth specified.

The function can be called from read-only cellviews also.

Arguments

d_cvId Cellview ID where shapes are to be searched.

?layerName *t_layerName*

Layer name on which shapes are to be searched.

?depth *x_depth*

Depth up to which shapes are to be searched in the hierarchy.

?createMarker *g_createMarker*

Specify whether markers are to be created. Valid values: `t` or `nil`.

?printReport *g_printReport*

Specify whether the report is to be printed in CIW.

Valid values: `t` or `nil`.

?showReportWindow *g_showReportWindow*

Specify whether the report is to be displayed in a separate window. Valid values: `t` or `nil`.

`?saveReport` *`t_saveReport`*

Specifies the directory where the report is to be saved. The report file follows the naming convention *`cellName`*`.rpt`.

`?markerSeverity` *`t_markerSeverity`*

Specifies the severity to be applied to the marker generated.

Value Returned

l_lists	Returns a list of lists. Each list displays (*t_layerName* (*t_type* ("sameCellView" *ld_fig*) ("differentClone" *ld_fig*) ("differentCellView" *ld_fig*)))
	When trim shapes and pathsegs are in the same cellview, they are returned in the *ld_fig* list of samecellView. When trim shapes and pathSegs are in a different clone, they are returned in the *ld_fig* list of differentClone. When trim shapes and pathsegs are in a different cellview, they are returned in the *ld_fig* list of differentCellView.
	The report returns the list of trim shapes that trim pathsegs on the specified layer name, at different levels in the hierarchy up to the specified depth, as type overlapping trim. It returns the list of pathsegs on the specified layer name that have abutting trim shapes at different levels in the hierarchy up to the specified depth, but do not have the relevant patch type set, as type missing patchType. If a trim shape has metal shapes abutted in the same cellview and also in a different cellview, the metal shapes only in the same cellview are reported.
	The trim shape is represented by a list, which could be a nested list, if it comes from more than one level down. Each level of the list has the ID of the instance in the corresponding level in the hierarchy. For example, if there is a trim shape in a master cellview two levels down, for the top cellview, it will return a nested list. The output of this function is same as dbGetTrueOverlaps.
	The function prints coordinates of the shapes.
nil	The report is not printed in CIW.

Examples

Prints the list of trim shapes in the hierarchy up to level 5.

```
leReportTrimmedShapesInCustomStyle( geGetEditCellView() ?depth 5 ?printReport t)
=> Metal1 shapes overlapped by a trim shape. Run converter to convert to Innovus-compatible style.
    Cellview cdn10FF:Level1TrimAndMetal1_2:layout - 2 found
        PathSeg at (84.322 -25.095) BBox : (77.379 -26.645) (91.265 -23.544)
```

```
        PathSeg at (65.869 -25.095) BBox : (59.572 -26.645) (72.166 -23.544)
    Cellview cdn10FF:test_InstsInside:layout - 8 found
        PathSeg at (-10.432 -49.682) BBox : (-18.500 -51.233) (-2.364 -48.133)
        PathSeg at (8.369 -48.432) BBox : (6.190 -49.983) (10.548 -46.883)
        PathSeg at (23.464 -41.408) BBox : (16.289 -42.408) (30.639 -40.408)
        PathSeg at (6.537 -41.678) BBox : (-0.019 -43.228) (13.092 -40.129)
        PathSeg at (22.891 -36.889) BBox : (15.868 -38.438) (29.914 -35.339)
        PathSeg at (-12.809 -32.072) BBox : (-21.163 -33.621) (-4.454 -30.521)
        PathSeg at (57.423 -2.660) BBox : (55.244 -4.210) (59.602 -1.111)
        PathSeg at (6.537 -36.416) BBox : (-0.019 -37.965) (13.092 -34.865)
    Cellview cdn10FF:Level1TrimAndMetal1:layout - 6 found
        PathSeg at (-38.075 -2.719) BBox : (-46.132 -3.719) (-30.018 -1.719)
        PathSeg at (84.186 -37.325) BBox : (76.129 -38.325) (92.243 -36.325)
        PathSeg at (84.935 -31.211) BBox : (76.942 -32.761) (92.928 -29.661)
        PathSeg at (51.242 -39.215) BBox : (48.020 -40.765) (54.463 -37.665)
        PathSeg at (70.133 -37.965) BBox : (67.954 -39.515) (72.312 -36.415)
        PathSeg at (47.069 -22.165) BBox : (40.484 -23.165) (53.653 -21.165)
    Cellview cdn10FF:testDesign2:layout - 1 found
        PathSeg at (-60.076 -23.187) BBox : (-73.506 -24.187) (-46.646 -22.187)
Metal1 shapes overlapped by a trim shape in a different clone. A manual conversion
is required.
    Cellview cdn10FF:test_InstsInside:layout - 6 found
        PathSeg at (76.314 -3.910) BBox : (73.093 -5.460) (79.536 -2.361)
        PathSeg at (41.788 4.364) BBox : (34.083 3.364) (49.493 5.364)
        PathSeg at (61.228 4.014) BBox : (54.931 2.465) (67.525 5.564)
        PathSeg at (43.759 8.883) BBox : (37.462 7.333) (50.056 10.434)
        PathSeg at (61.228 10.091) BBox : (54.931 8.540) (67.525 11.640)
        PathSeg at (80.487 13.140) BBox : (73.903 12.140) (87.072 14.140)
Metal1 shapes overlapped by a trim shape in a different cellview. A manual
conversion is required.
    Cellview cdn10FF:Level1TrimAndMetal1:layout - 1 found
        PathSeg at (-37.326 3.395) BBox : (-45.319 1.845) (-29.333 4.945)
    Cellview cdn10FF:testDesign2:layout - 1 found
        PathSeg at (-26.700 8.913) BBox : (-38.451 7.913) (-14.949 9.913)
Trim shapes are not abutted to any Metal1 shape in the same cellview. Delete the
trim shape or create a metal shape manually.
    Cellview cdn10FF:Level1TrimAndMetal1_2:layout - 4 found
        Trim at (69.519 -37.947) BBox : (66.562 -39.491) (72.476 -36.404)
        Trim at (51.957 -40.788) BBox : (51.007 -47.338) (52.907 -34.238)
        Trim at (69.806 -31.194) BBox : (68.856 -32.737) (70.756 -29.650)
        Trim at (48.450 -24.555) BBox : (45.498 -31.105) (51.401 -18.005)
```

```
    Cellview cdn10FF:Level1TrimAndMetal1:layout - 2 found
        Trim at (-52.309 9.435) BBox : (-53.259 8.088) (-51.359 10.783)
        Trim at (69.952 -25.171) BBox : (69.002 -26.518) (70.902 -23.823)
(("Metal1"
    (("overlapping trim"
        (("sameCellView"
            (db:0x248fcb9f db:0x248fc020 db:0x248fc01c db:0x248fc01e db:0x248fc01d
            db:0x248fc533 db:0x248fc53a db:0x248fc531 db:0x248fc534 db:0x248fc537
            db:0x248fc536 db:0x248fc535 db:0x248fc532 db:0x248fcba0 db:0x248fcba1
            db:0x248fd39e db:0x248fd39f
            )
        )
        ("differentClone"
            (db:0x248fc51e db:0x248fc539 db:0x248fc538 db:0x248fc51d db:0x248fc51c
            db:0x248fc51f
            )
        )
        ("differentCellView"
            ((db:0x248fd09b db:0x248fcba0) db:0x248fd39e)
            )
        )
    )
        ("missing same cellview metal"
            (("sameCellView"
                (db:0x248fcb9f db:0x248fc020 db:0x248fc01f db:0x248fc01e
db:0x248fc01c)
            )
            )
        )
        )
)
)
```

leResizeLSW

```
leResizeLSW(
     l_bBox
     )
     => t / nil
```

Description

Sets the Layer Selection Window's (LSW) bounding box to *l_bBox*. The bounding box specifies the lower left and upper right corners of the LSW window.

Note: This SKILL function is not supported in Palette assistants. On execution of this SKILL function, the following warning message is displayed.
WARNING (LE-105566): The SKILL function leResizeLSW cannot be used because the LSW is disabled. The Palette assistants are enabled as replacements for the LSW. To enable LSW, set the UNIX shell environment variable CDS_PALETTE_OFF before launching Virtuoso.

Arguments

l_bBox	List of two window coordinates specifying the lower left corner and the upper right corner of the LSW.

Value Returned

t	The LSW's bounding box has changed.
nil	The LSW's bounding box has not changed.

Example

Moves the LSW lower left corner to 20:20 and stretches the upper right corner to 220:820. The resulting LSW, excluding the border and window title, is 200 pixels wide and 800 pixels high.

```
leResizeLSW( list( 20:20 220:820) )
```

leSetAllGridObjectsVisible

```
leSetAllGridObjectsVisible(
     g_isVisible
     [ d_techFileId ]
     )
     => t
```

Description

Sets all LSW grid pane objects to visible if *g_isVisible* is `t`, or invisible if *g_isVisible* is `nil`.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isVisible	Sets all grid objects to visible if `t`, or invisible if `nil`.
d_techFileId	The technology file ID.

Value Returned

`t`	Objects are set to visible or invisible.

leSetAllLayerSelectable

```
leSetAllLayerSelectable(
     g_isSelectable
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the selectability of all layers in technology file *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isSelectable	Specifies whether all layers in library *d_techFileId* are selectable. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file containing the layers.

Value Returned

`t`	The selectability of all layers is set to the specified value.
`nil`	The selectability of all layers is not set to the specified value.

Example

Sets all layers in technology file `myTechFile` to be selectable.

```
leSetAllLayerSelectable( t myTechFile )
```

leSetAllLayerValid

```
leSetAllLayerValid(
     g_isValid
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the validity of all layers in technology file *d_techFileId* as specified by *g_isValid*. If *d_techFileId* is not specified, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isValid	Specifies whether the layers in technology file *d_techFileId* are valid. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file containing the layers.

Value Returned

`t`	The layers are set to the specified validity.
`nil`	The layers are not set to the specified validity.

Example

Sets all layers in the technology file `myTechFile` to be valid.

```
leSetAllLayerValid( t myTechFile)
```

leSetAllLayerVisible

```
leSetAllLayerVisible(
     g_isVisible
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the visibility of all layers in technology file *d_techFileId* as specified by *g_isVisible*. If *d_techFileId* is not specified, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isVisible	Specifies whether all layers in technology file *d_techFileId* are visible. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file containing the layers.

Value Returned

`t`	The visibility of all layers is set to the specified value.
`nil`	The visibility of all layers is not set to the specified value.

Example

Sets all layers in library `myTechFile` to be visible and returns `t`.

```
leSetAllLayerVisible( t myTechFile )
```

leSetAllObjectsSelectable

```
leSetAllObjectsSelectable(
     g_isSelectable
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the selectability of LSW objects in technology file *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not given, the current LSW technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isSelectable	Whether to turn on (t), or off (nil) the selectability. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file.

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

Sets LSW objects in technology file `myTechFile` to be selectable.

```
leSetAllObjectsSelectable( t myTechFile )
```

leSetAllObjectsVisible

```
leSetAllObjectsVisible(
     g_isVisible
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the visibility of LSW objects in technology file *d_techFileId* as specified by *g_isVisible*. If *d_techFileId* is not given, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isVisible	Whether to turn on (`t`), or off (`nil`) the visibility. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file containing the layers.

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

Sets all LSW objects in technology file my`TechFile` to be visible.

```
leSetAllObjectsVisible( t myTechFile )
```

leSetAreaEstimatorParameters

```
leSetAreaEstimatorParameters(
     l_functionParamList
     )
     => l_functionParamList / nil
```

Description

Opens a form to update the parameter list to new values. On pressing *OK* on the form, the updated parameter list is returned.

Arguments

l_functionParamList

List specifying a simple display name for a cyclic list in the graphical user interface.

Value Returned

l_funcParamList	Returns *l_funcParamList* if the function is successful.
nil	Returns nil if the function is not successful.

leSetEditFigGroup

```
leSetEditFigGroup(
     d_figGroupId
     [ w_windowID ]
     )
     => t / nil
```

Description

Sets the Edit-In-Place context for editing a specific group within a nested set of groups. For example, when `fgBottom` is contained in `fgMiddle`, and `fgMiddle`, is contained in `fgTop`, you can use this function and the figGroup ID for `fgBottom` to start Edit-in-Place for `fgBottom`.

Arguments

d_figGroupId	Database ID of the figure group for which you want to Edit-In-Place.
w_windowId	Database ID of the window containing the figure group. If you do not specify *w_windowId*, the current window is used.

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

For the specified *windowID*, sets the Edit-In-Place context for the specified *figGroupId*.

```
leSetEditFigGroup(figGroupId windowId)
```

leSetEntryLayer

```
leSetEntryLayer(
     l_layerPurposePair
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the entry layer for technology file *d_techFileId* to the layer specified by *l_layerPurposePair*, which can be either a list containing a layer name and a layer purpose, or just a layer name (the layer purpose then defaults to `drawing`). If *d_techFileId* is not specified, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

l_layerPurposePair

Specifies the layer to use as the entry layer for library *d_techFileId*.
Valid Values: any valid layer name and layer purpose

d_techFileId — Database ID for the technology file specified.

Value Returned

`t`	The entry layer is set; otherwise.
`nil`	The entry layer is not set and the software generates a warning message.

Example

Makes `poly1` with the purpose `drawing` the entry layer for the technology file `myTechFile` and returns `t`.

```
leSetEntryLayer( list("poly1" "drawing") myTechFile )
```

leSetFocusToEditableFieldsInStatusToolbar

```
leSetFocusToEditableFieldsInStatusToolbar(
     w_windowId
     )
     => t / nil
```

Description

Sets the focus to the editable *X* field on the Status toolbar of the window, *w_windowId.*

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command was successful.
nil	The command was unsuccessful.

Example

Sets the focus to the editable X field of status toolbar of the current window.

```
leSetFocusToEditableFieldsInStatusToolbar(hiGetCurrentWindow())
```

leSetFormSnapMode

```
leSetFormSnapMode(
     t_SnapMode
     )
     => t_SnapMode
```

Description

Sets the cursor snap mode in a valid active window.

Arguments

t_SnapMode	Specifies the cursor snap mode. Valid Values: `diagonal`, `orthogonal`, `L90XFirst`, `L90YFirst`, `anyAngle`

Value Returned

t_SnapMode	Returns the cursor snap mode.

Example

Sets the cursor snap mode and returns the snap mode set `"diagonal"`.

```
leSetFormSnapMode("diagonal")
```

leSetInstSelectable

```
leSetInstSelectable(
     g_isSelectable
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the selectability of instances in the technology file specified by *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isSelectable	Specifies if the instances are selectable. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file.

Value Returned

`t`	The selectability of the instances is set to the specified value.
`nil`	The selectability of the instances is not set to the specified value.

Example

Sets instances in `myTechFile` to be selectable and returns `t`.

```
leSetInstSelectable( t myTechFile )
```

leSetLayerAttributes

```
leSetLayerAttributes(
     d_techFileId
     g_layerPurposePair
     l_attributeList
     )
     => t / nil
```

Description

Sets the attributes of the specified layer in the technology file *d_techFileId* as specified by *g_layerPurposePair* and *l_attributeList*.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

d_techFileId	The database ID of the technology file containing the layer.
g_layerPurposePair	Specifies the layer in the technology file whose attributes you change.
l_attributeList	Specifies attributes of the layer in the technology file *d_techFileId*. These attributes occur in the following order: *g_isValid*, *g_isSelectable*, *g_isVisible*. Only valid layers are displayed in the Layer Selection Window. A selectable layer must also be valid and visible. Valid Values: `t` or `nil`

Value Returned

`t`	The attributes of the layer are set to the specified values.
`nil`	The attributes of the layer are not set to the specified values.

Example

Sets layer `metal1Res drawing` in `techFileId` to valid, not selectable, and visible.

```
leSetLayerAttributes(techFileId list('("metal1Res" "drawing") t nil t))
```

leSetLayerSelectable

```
leSetLayerSelectable(
     l_layerPurposePair
     g_isSelectable
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the selectability of shapes on layer *l_layerPurposePair* in technology file *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used. For shapes on a layer to be selectable, the layer must be valid, visible, and selectable.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

l_layerPurposePair

Layer-purpose pair whose selectability is set.

g_isSelectable

Specifies if the layer *l_layerPurposePair* is selectable.
Valid Values: `t` or `nil`

d_techFileId

The database ID of the technology file.

Value Returned

`t`

The selectability of shapes is set.

`nil`

The selectability of shapes is not set.

Examples

Any of these examples sets layer `poly1` purpose `drawing` to be selectable and returns `t`.

```
leSetLayerSelectable( list("poly1" "drawing") t )
leSetLayerselectable( list("poly1") t )
leSetLayerSelectable( "poly1" t )
```

leSetLayerValid

```
leSetLayerValid(
     l_layerPurposePair
     g_isValid
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the validity of layer *l_layerPurposePair* as specified by *g_isValid*. If *d_techFileId* is not specified, the current technology file is used. Only valid layers are displayed in the Layer Selection Window.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

l_layerPurposePair

Layer-purpose pair whose validity you want to set.

g_isValid

Specifies whether the layer *l_layerPurposePair* is valid. Valid Values: `t` or `nil`

d_techFileId

The database ID of the technology file containing the layer-purpose pair.

Value Returned

`t`

The validity of the layer-purpose pair is set.

`nil`

The validity of the layer-purpose pair is not set.

Example

Sets the layer-purpose pair `metal1 drawing` in technology file `myTechFile` to be invalid.

```
leSetLayerValid( list("metal1" "drawing") nil myTechFile )
```

leSetLayerVisible

```
leSetLayerVisible(
     l_layerPurposePair
     g_isVisible
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the visibility of the layer specified by the layer-purpose pair as specified by *g_isVisible*. If *d_techFileId* is not specified, the current technology file Layer Selection Window (LSW) is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

l_layerPurposePair

Layer-purpose pair to set visible.

g_isVisible	Specifies whether the layer is visible.Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file.

Value Returned

`t`	The visibility of the layer is set to the specified value.
`nil`	The visibility of the layer is not set to the specified value.

Examples

Any of these examples sets layer `poly1` purpose `drawing` to be visible and returns `t`.

```
leSetLayerVisible( list("poly1" "drawing") t )
leSetLayerVisible( list("poly1") t )
leSetLayerVisible( "poly1" t )
```

leSetLSWFilter

```
leSetLSWFilter(
     l_filters
     )
     => t / nil
```

Description

Specifies which layer-purpose pairs in the technology database to display in the Layer Selection Window (LSW). You can include one or more of the following types of layers: valid layers, rule layers, user layers.

Arguments

l_filters	List of string values, enclosed in double quotation marks, specifying the types of layers that appear in the LSW. You must specify at least one value. - When set to `"valid"`, specifies layers listed as valid in the `techDisplays` subsection of the `layerDefinitions` section. - When set to `"rules"`, specifies layers listed in the `leLswLayers` subsection of the `leRules` section. - When set to `"user"`, specifies user-defined layers listed in the `techLayers` subsection of the `layerDefinitions` section. Valid Values: "`valid`", "`rule`", "`user`"

Value Returned

`t`	The filters were applied successfully.
`nil`	The filters were not applied successfully.

Example

The "`rule`" and "`user`" types of layers appear in the LSW.

```
leSetLSWFilter(list("rule" "user"))
```

leSetObjectSelectable

```
leSetObjectSelectable(
     t_objectName
     g_isSelectable
     )
     => t / nil
```

Description

Turns on or off the selectability of the layers used for *objectName* or updates the selectability of instances or pins as specified by *g_isSelectable*.

Arguments

t_objectName	The type of object affected. Valid Values: `"Instances", "Pins", "Vias", "Label",` Shapes `("Circle/Ellipse", "Donut", "Path", "PathSeg", "Polygon", "Rectangle", "Other Shapes"), "Mosaic",` Markers `("Acknowledge Warning", "Annotation", "Critical Error", "Error", "Fatal", "Info", "Signed Off Critical Error", "Signed Off Error", "Warning"), "Fluid Guardring", "Fluid Shapes", "Fig Groups",` Boundaries `("P&R Boundary", "Area Boundary", "Snap Boundary", "Cluster Boundary"),` Soft Blocks `("SoftBlock Pin", "SoftBlock P&R", "SoftBlock Snap"),` Blockages `("Halo Blockage", "Placement Blockage", "Routing Blockage", "Fill Blockage", "Slot Blockage", "Pin Blockage", "Feedthru Blockage", "Screen Blockage"), "Symmetric Axes",` Rows `("Row", "Mixed Row", "Custom Placement Area")`
g_isSelectable	Whether to turn on (`t`), or off (`nil`) the selectability. Valid Values: `t` or `nil`

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

Sets instances selectable.

```
leSetObjectSelectable( "inst" t )
```

leSetObjectVisible

```
leSetObjectVisible(
     t_objectName
     g_isVisible
     )
     => t / nil
```

Description

Turns on or off the visibility of the layers used for *objectName* or updates the visibility of instances or pins as specified by *g_isVisible*.

Arguments

t_objectName	The type of object affected. Valid Values: `"Instances", "Pins", "Vias", "Label",` Shapes `("Circle/Ellipse", "Donut", "Path", "PathSeg", "Polygon", "Rectangle", "Other Shapes"), "Mosaic",` Markers `("Acknowledge Warning", "Annotation", "Critical Error", "Error", "Fatal", "Info", "Signed Off Critical Error", "Signed Off Error", "Warning"), "Fig Groups",` Boundaries `("P&R Boundary", "Area Boundary", "Snap Boundary", "Cluster Boundary"),` Blockages `("Halo Blockage", "Placement Blockage", "Routing Blockage", "Fill Blockage", "Slot Blockage", "Pin Blockage", "Feedthru Blockage", "Screen Blockage"), "Symmetric Axes",` Rows `("Row", "Mixed Row", "Custom Placement Area"), "Tracks",` Grids `("Placement Grid", "Routing Grid"),"Snap Patterns"`
g_isVisible	Whether to turn on (`t`), or off (`nil`) the visibility. Valid Values: `t` or `nil`

Value Returned

`t`	The operation succeeded.
`nil`	The operation failed.

Example

Sets the instances visible.

```
leSetObjectVisible( "inst" t )
```

leSetPinSelectable

```
leSetPinSelectable(
     g_isSelectable
     [ d_techFileId ]
     )
     => t / nil
```

Description

Sets the selectability of pins in the technology file specified by *d_techFileId* as specified by *g_isSelectable*. If *d_techFileId* is not specified, the current technology file is used.

Note: In Palette mode, this SKILL API will work only for the current active window.

Arguments

g_isSelectable	Specifies if the pins in library *d_techFileId* are selectable. Valid Values: `t` or `nil`
d_techFileId	The database ID of the technology file.

Value Returned

`t`	The selectability of the pins is set to the specified value.
`nil`	The selectability of the pins is not set to the specified value.

Example

Sets pins in `myTechFile` to be selectable and returns `t`.

```
leSetPinSelectable( t myTechFile )
```

leSetStopLevelToMaxHierDepth

```
leSetStopLevelToMaxHierDepth(
     [ w_windowId ]
     )
     => x_maxdepth / nil
```

Description

Sets the display stop level to the maximum hierarchy depth, which is 31.

Arguments

w_windowId	ID of the window for which the display stop level is set to maximum. If you do not specify a window ID, the display level is set for the current window.

Value Returned

x_maxdepth	Returns the maximum hierarchy depth set for the display stop level.
nil	The function is not successful.

Example

Sets the maximum hierarchy depth for the display stop level in the current window and returns the value as 31.

```
leSetStopLevelToMaxHierDepth()
```

leToggleAutoZoomPan

```
leToggleAutoZoomPan(
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the current value of dynamic zoom for the specified *w_windowId*. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window containing the selected objects.

Value Returned

t	Dynamic zoom is toggled.
nil	Dynamic zoom is not toggled.

Example

Turns dynamic zoom off if it was previously on, or on if it was previously off.

```
leToggleAutoZoomPan()
```

leToggleGravity

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

Description

Turns gravity on or off.

Arguments

None

Value Returned

`t`	Gravity is toggled.
`nil`	Gravity is not toggled.

Example

Turns gravity off if it was previously on, or on if it was previously off.

```
leToggleGravity()
```

leToggleKeepFirstName

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

Description

Toggles the *Keep First Name* option on or off on the Create Pin form. When this option is on, you can create multiple pins with the same terminal name.

Arguments

None

Values Returned

`t`	Returned when the option is on
`nil`	Returned when the option is off

Example

```
leToggleKeepFirstName()
```

leToggleMagnifier

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

Description

Toggles the visibility of the magnifier between on and off.

Arguments

None

Value Returned

`t`	The magnifier visibility toggles successfully between on and off.
`nil`	The magnifier visibility does not toggle.

Examples

If the magnifier is on, this function turns it off. If the magnifier is off, this function displays it based on the current settings in the Magnifier Options form.

```
leToggleMagnifier()
```

leToggleMaintainConnections

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

Description

Turns Maintain Connections on or off.

Arguments

None

Value Returned

`t`	Maintain Connections is toggled.
`nil`	Maintain Connections is not toggled.

Example

Turns Maintain Connections off if it was previously on, or on if it was previously off.

```
leToggleMaintainConnections()
```

leToggleRuleGravity

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

Description

Turns rule gravity on or off.

Arguments

None

Value Returned

`t`	Rule gravity is toggled.
`nil`	Rule gravity is not toggled.

Example

Turns rule gravity off if it was previously on, or on if it was previously off.

```
leToggleRuleGravity()
```

leToggleSmartSnap

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

Description

Toggles the smart snap mode of ruler between no snapping (off) and snap to any target (on).

Arguments

None

Value Returned

`t`	The smart snapping toggles successfully between on and off.
`nil`	The smart snapping does not toggle.

Examples

If smart snapping is on, this function turns it off. If smart snapping is off, this function turns it on and sets the ruler to snap to any target.

```
leToggleSmartSnap()
```

leUnregisterUseGravity

```
leUnregisterUseGravity(
     t_cmdName
     )
     => t / nil
```

Description

Prevents a user-defined SKILL enter function to use gravity. By default, gravity is available only for SKILL enter functions for which gravity has been enabled using the `leRegisterUseGravity` function.

Arguments

t_cmdName	The name of the command for which you want to stop gravity during entering of points.

Value Returned

`t`	function name was registering for using gravity and has now been unregistered
`nil`	the named function was not registered for using gravity.

Example

For the following user-defined SKILL enter function:

```
(defun MyStretchSpecial (...)
  ...
  (enterPoint ?cmdName "StretchSpecial" ...)
  ) ; defun
```

If you want to use the gravity feature for the function `StretchSpecial` when the command runs, use the following function:

```
(leRegisterUseGravity "StretchSpecial") ; Gravity now works for StretchSpecial
(MyStretchSpecial ...)
```

For as long as `MyStretchSpecial` function runs, the entered points will use gravity as per the Editor Options values. When the function no longer needs to use gravity, use the following function:

```
(leUnregisterUseGravity "StretchSpecial") ; Gravity will not work for
StretchSpecial
```

leUnregUserLayerSelectionFilter

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

Description

Unregisters user defined layer selection filter.

Arguments

None

Value Returned

`t`	Returns `t` if the user defined layer selection filter is successfully unregistered.
`nil`	Returns `nil` if the user defined layer selection filter is not successfully unregistered.

Example

Returns `t` if the user defined layer selection filter is successfully unregistered and `nil` if it is not.

```
leUnregUserLayerSelectionFilter()
```

leUnregUserObjectSelectionFilter

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

Description

Unregisters user defined object selection filter.

Arguments

None

Value Returned

`t`	Returns `t` if the user defined object selection filter is successfully unregistered.
`nil`	Returns `nil` if the user defined object selection filter is not successfully unregistered.

Example

Returns `t` if the user defined object selection filter is successfully unregistered. and `nil` if it is not.

```
leUnregUserObjectSelectionFilter()
```

leUnRegAreaEstimator

```
leUnRegAreaEstimator(
     t_functionAlias
     l_legalObjectTypes
     )
     => t / nil
```

Description

Unregisters a particular area estimation function for one or more object types.

Arguments

t_functionAlias	Text string specifying a simple display name for a cyclic list in the graphical user interface.
l_legalObjectTypes	List of legal object types for which this function will work. Valid Values: `'cellview`, `'cluster`, `'group`, `'inst`

Value Returned

`t`	Returns `t` if the function alias name is successfully unregistered.
`nil`	Returns `nil` if the function alias name is not successfully unregistered.

Example

Unregisters `simpleAreaEst` as the alias name which appears in the cyclic field of the Area Estimator field for the `cluster` object type.

```
leUnRegAliasEstimator( "simpleAreaEst" ('cluster) )
```

leUnRegClusterBdyEstimator

```
leUnRegClusterBdyEstimator(
     t_displayName
     )
     => t / nil
```

Description

Unregisters the name of the estimator which will subsequently no longer appear in the cyclic field of the *Area Estimator* field in the Create Cluster Boundary form in Layout XL.

Arguments

t_displayName	Name that appears in the *Area Estimator* field in the Create Cluster Boundary form in Layout XL

Value Returned

`t`	Returns `t` if the area estimator is successfully unregistered.
`nil`	Returns `nil` if the area estimator is not successfully unregistered.

Example

Unregisters `estimateFunc` as the name that appears in the *Area Estimator* field in the Create Cluster Boundary form in Layout XL.

```
leUnRegClusterBdyEstimator("estimateFunc")
```

leZoomToPoint

```
leZoomToPoint(
     w_windowId
     l_points
     )
     => t / nil
```

Description

Places the cursor in the window, *w_windowId,* at the coordinates specified in *l_points*.

Arguments

w_windowId	Window ID of the window specified.
l_points	List of two XY coordinates.

Value Returned

t	The cursor is placed at the coordinates in the window specified.
nil	The cursor is not placed at the coordinates in the window specified.

Example

```
leZoomToPoint(hiGetCurrentWindow(), 100:0)
```

leZoomToSelSet

```
leZoomToSelSet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Zooms to fit the selected objects in the specified window. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window containing the selected objects.

Value Returned

t	The system successfully zooms the display to selected objects in the specified window.
nil	The system does not successfully zoom the display to selected objects in the specified window.

Reference Point Functions

You can use reference point functions to manipulate reference points and the cursor. The functionality does not appear on banner menus.

A reference point is a location set by the user and maintained by the editor. Reference points are used to measure relative distances, such as the distance from the reference point to the cursor.

Each cellview can have only one reference point. The reference point can be active or inactive. You can display a star-marker at the location of the reference point by setting the `displayRefPoint` global environment variable. Reference points are displayed on the `hilight drawing3` layer.

Reference Point Display: 

You can set the reference point coincident with a particular edge or vertex using `gravityOn`, then measure the distance from the reference point to the cursor as a means of separating objects by design rule spacings. You can also set the reference point automatically whenever you enter a point by setting the `autoSetRefPoint` global environment variable. The distance from the cursor to the reference point is displayed in the status banner of the window.

You can move the cursor using bindkeys rather than the mouse. You can move the cursor to the reference point, then “bump” the cursor by fixed distances (for example, 0.5 microns, 1.0 microns) and enter a point using a bindkey. In this way, accurate distances can be measured and created at “high altitude.”

Related Topics

leGetRefPoint

leIsRefPointActive

leMoveCursor

leMoveCursorToRefPoint

leSetRefPoint

leSetRefPointInactive

leGetRefPoint

```
leGetRefPoint(
     d_cellViewId
     )
     => l_point / nil
```

Description

Returns the reference point of cellview *d_cellViewId*. The reference point must be active.

Arguments

d_cellViewId	The database ID of the cellview containing the reference point.

Value Returned

l_point	Returns the reference point.
nil	Does not return the reference point.

Example

Returns the coordinates of the reference point in the cellview cell2.

```
leGetRefPoint( cell2 )
```

leIsRefPointActive

```
leIsRefPointActive(
     d_cellViewId
     )
     => t / nil
```

Description

Checks whether the cellview *d_cellViewId* contains an active reference point.

Arguments

d_cellViewId	The database ID of the cellview containing the reference point.

Value Returned

t	The reference point is active for cellview *d_cellViewId*.
nil	The reference point is not active for cellview *d_cellViewId*.

Example

Returns t if the reference point is active in the cellview cell2.

```
leIsRefPointActive( cell2 )
```

leMoveCursor

```
leMoveCursor(
     f_dX
     f_dY
     )
     => t / nil
```

Description

Moves the cursor in the current window a relative distance specified by *f_dX* and *f_dY*.

Arguments

f_dX	Distance to move the cursor in the X direction.
f_dY	Distance to move the cursor in the Y direction.

Value Returned

t	The cursor moves.
nil	The cursor does not move.

Example

Moves the cursor 5.5 in the X direction and 10.5 in the Y direction and returns t.

```
leMoveCursor( 5.5 10.5 )
```

leMoveCursorToRefPoint

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

Description

Moves the cursor to the reference point of the cellview contained in the current window.

Arguments

None

Value Returned

`t`	The reference point is active.
`nil`	The reference point is not active and the software generates an error dialog box.

Example

Moves the cursor to the location of the reference point and returns `t`.

```
leMoveCursorToRefPoint()
```

leSetRefPoint

```
leSetRefPoint(
     d_cellViewId
     l_point
     )
     => t / nil
```

Description

Sets the reference point of cellview *d_cellViewId* to the point *l_point*. The reference point is set to be active if it was inactive.

Arguments

d_cellViewId	The database ID of the cellview containing the reference point.
l_point	Coordinate specified as the reference point.

Value Returned

`t`	The reference point is set.
`nil`	The reference point is not set and the software generates a warning message.

Example

Locates the reference point of the cellview `cell2` at the coordinate 25:25 and returns `t`.

```
leSetRefPoint( cell2 25:25 )
```

leSetRefPointInactive

```
leSetRefPointInactive(
     d_cellViewId
     )
     => t / nil
```

Description

Marks the reference point for cellview *d_cellViewId* as inactive. To make a reference point active again, use `leSetRefPoint`.

Arguments

d_cellViewId	The database ID of the cellview containing the reference point.

Value Returned

`t`	The reference point is inactive.
`nil`	The reference point is not inactive.

Example

Makes the reference point in the cellview `cell2` inactive and returns `t`.

```
leSetRefPointInactive( cell2 )
```

Capture and Replay Assistant Functions

You can record a series of actions and replay them in your designs when required. This helps you to perform repetitive tasks quickly and consistently across cellviews and sessions. You can use the following SKILL functions to record and replay your design actions in Layout EXL and higher tiers:

- leCARCommand
- leCARStartCapture
- leCARStopCapture

leCARCommand

```
leCARCommand(
     t_replayFileName
     [ g_enterPointFlag ]
     [ g_debugFlag ]
     )
     => l_recordedActions / nil
```

Description

(Layout EXL and higher tiers only) Replays the set of actions recorded in a `.car` file.

Arguments

t_replayFileName	Specifies the name of the file to be replayed. You need enter only the file name; the `.car` extension and the absolute path are not required.
g_enterPointFlag	A Boolean value indicating whether to run the replay file with a user-specified point or with the starting point saved in the recording. ■ `t`: Runs the replay with the user-specified point. ■ `nil`: Runs the replay at the exact same location as specified in the recording. The default value is `nil`.
g_debugFlag	A Boolean value specifying whether to display the content of the replay file in the log file. This is for testing purposes. The default value is `nil`.

Value Returned

l_recordedActions	Returns the series of actions that are performed while replaying the specified replay file.
nil	Recording could not be replayed successfully.

Example

The following example returns the series of actions that are performed while replaying the file createRectangle. The actions are performed at the points saved in the replay file.

```
leCARCommand("createRectangle")
t
lePointX = 1.472000
1.472
lePointY = 4.842500
4.8425
;
; ============================================================================ ;
;  FileName: "./.cadence/dfII/captureReplay/createRectangle.car"
;  Date: Feb  9 11:20:00 2023
;  Author: savt
;  Command Name: createRectangle
;  Command Description: Creates a rectangle
;  Group Name: Macros1
; ============================================================================ ;
;  Recorded part:
t
;  Recorded part:
t
```

The following example returns the series of actions performed while replaying the file createRectangle. The actions are performed at the user-specified point.

```
leCARCommand("createRectangle" t)
(-5.034 4.071)
lePointX = -5.034000
-5.034
lePointY = 4.071000
4.071
t
;
```

```
; ============================================================================ ;
;  FileName: "./.cadence/dfII/captureReplay/createRectangle.car"
;  Date: Feb  9 11:20:00 2023
;  Author: savt
;  Command Name: createRectangle
;  Command Description: Creates a rectangle
;  Group Name: Macros1
; ============================================================================ ;
;  Recorded part:
```

Related Topics

leCARStartCapture

leCARStopCapture

Capture and Replay Design Actions

Replaying Your Recorded Actions in a Design

leCARReloadReplays

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

Description

(Layout EXL and higher tiers only) Reloads the replays in all the open sessions. The replays with `.car` extension are displayed in the refreshed list of replay buttons.

Arguments

None

Value Returned

`t`	Replays reloaded successfully in all the open sessions.
`nil`	Replays could not be reloaded successfully in all the open sessions.

Example

Reloads the replays in all the open sessions.

```
leCARReloadReplays()
t
```

Related Topics

leCARStopCapture

leCARCommand

Capture and Replay Design Actions

Recording Your Actions in a Design

leCARStartCapture

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

Description

(Layout EXL and higher tiers only) Starts recording the actions that you perform in the current design window.

Arguments

None

Value Returned

`t`	Recording started successfully.
`nil`	Recording could not be started.

Example

Starts recording your actions in the current design window.

```
leCARStartCapture()
t
```

Related Topics

leCARStopCapture

leCARCommand

Capture and Replay Design Actions

Recording Your Actions in a Design

leCARStopCapture

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

Description

(Layout EXL and higher tiers only) Stops the recording that was started using the `leCARStartCapture` SKILL function.

When you run `leCARStopCapture`, the Save Recording form is displayed, where you can specify details such as the command name, description, group name, and location of the replay file.

Arguments

None

Value Returned

`t`	Recording stopped successfully.
`nil`	Recording could not be stopped.

Example

Stops recording actions in a design window.

```
leCARStopCapture()
t
```

Related Topics

leCARStartCapture

leCARCommand

Capture and Replay Design Actions

Recording Your Actions in a Design

Save Recording Form

Measurement Functions

Measurement functions enable you to manage measurements and rulers. For example, you can create a ruler with specific coordinates in a cellview. You can also clear the measurements in a hierarchy in a specified window.

Related Topics

leClearAllMeasurement

leCreateMeasurement

leHiClearMeasurement

leHiClearMeasurementInHier

leHiCreateMeasurement

leClearAllMeasurement

```
leClearAllMeasurement(
     d_cellViewId ]
     )
     => t
```

Description

Deletes all measurements in the specified cellview.

Arguments

d_cellViewId	The database ID of the cellview containing the instances you want to re-master.

Value Returned

`t`	The command is successful.
`nil`	The command is not successful.

Example

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
      leClearAllMeasurement(cellView)
  )
)
```

leCreateMeasurement

```
leCreateMeasurement(
     d_cellViewId
     l_points
     [ g_saveRulers ]
     [ t_rulerDisplayType ]
     )
     => d_rulerID / nil
```

Description

Creates a measurement in cellview *d_cellViewId* with the coordinates listed in *l_points*.

Arguments

d_cellViewId	The database ID of the cellview containing the measurement.
l_points	List of coordinates entered to create the measurement.
g_saveRulers	Specifies whether the measurement is savable or not. The default value is `t`. Valid Values: `t` or `nil`
t_rulerDisplayType	Specifies the display type of the measurement. The default value is `Ruler`. Valid Values: `Ruler`, `Distance`, or `Both`.

Value Returned

d_rulerID	The database ID of the measurement.
`nil`	The measurement is not created.

Example

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
     leCreateMeasurement(cellView list('(0 10) '(10 10)))
  )
)
```

leHiClearMeasurement

```
leHiClearMeasurement(
     [ w_windowId ]
     )
     => t
```

Description

Clears the edit cellview measurement for the specified window. If `windowId` is not specified, the current window is used.

Arguments

w_windowId	ID of the window.

Value Returned

`t`	The command is successful.

Example

```
when(window = hiGetCurrentWindow()
   leHiClearMeasurement(window)
)
```

leHiClearMeasurementInHier

```
leHiClearMeasurementInHier(
     [ w_windowId ]
     [ g_openAsEdit ]
     )
     => t
```

Description

Clears the measurements in the hierarchy in the specified window. If `windowId` is not specified, the current window is used.

Arguments

w_windowId	ID of the window.
g_openAsEdit	Opens a cellview in edit mode to delete rulers. When set to `nil`, rulers are deleted without opening the cellview.

Value Returned

`t`	The command is successful.

Example

The following example deletes rulers in the current window without opening cellviews in edit mode.

```
leHiClearMeasurementInHier(hiGetCurrentWindow() nil)
```

leHiCreateMeasurement

```
leHiCreateMeasurement(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the create measurement enter function in the edited cellview of the specified window. If `windowId` is not specified, the current window is used.

Arguments

w_windowId	ID of the window.

Value Returned

`t`	The command is successful.
`nil`	The command is not successful.

Example

```
when(window = hiGetCurrentWindow()
  leHiCreateMeasurement(window)
)
```

Info Balloon Functions

Information balloons displays information about an object when the mouse cursor is placed over the object. You can use SKILL functions to perform tasks such as turn the information balloon on or off, open the Info Balloon form, or cycle through overlapping objects displaying Info Balloon details for individual objects.

Related Topics

leBalloonCycleThru

leBalloonToggleOnOff

leHiEditBalloonOptions

leHiEditObjectInfo

leBalloonCycleThru

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

Description

Cycles through overlapping objects displaying *Info Balloon* information for the individual objects. By default, the function is registered to *Ctrl + i*.

Arguments

None

Value Returned

`t`	The command is successful.
`nil`	The command is not successful.

Example

```
leBalloonCycleThru()
```

Returns `t`.

leBalloonToggleOnOff

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

Description

Turns Info Balloons on or off. Info Balloons opens a window which displays information about an object when the mouse cursor is placed over the object.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Example

```
leBalloonToggleOnOff()
```

leHiEditBalloonOptions

```
leHiEditBalloonOptions(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Info Balloon form.

Arguments

w_windowId	Window ID of the window to use.

Value Returned

t	The command was successful.
nil	The command was unsuccessful.

Example

```
leHiEditBalloonOptions()
```

leHiEditObjectInfo

```
leHiEditObjectInfo(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the *Setup* tab of the *Dynamic Display* form. The form allows you to customize data displayed by Info Balloon and Dynamic Measurement.

Arguments

w_windowId	Window ID of the window to use.

Value Returned

t	The command was successful.
nil	The command was unsuccessful.

Example

```
leHiEditObjectInfo()
```

Search and Replace Functions

You can use SKILL functions to locate specific types of objects in a design. You can replace these objects with objects of a different type or with different criteria. For example, you can change the master library, cell, or view name for an instance.

Related Topics

leRemasterInstances

leReplace

leReplaceAnyInstMaster

leSearchHierarchy

leRemasterInstances

```
leRemasterInstances(
     d_cellViewId
     t_searchLibrary
     t_searchCellview
     t_searchViewName
     t_updateLibrary
     t_updateCellview
     t_updateViewName
     [ g_checkTerminals ]
     )
     => t / nil
```

Description

Searches for instances based on the view name, cell name, and the library name for a given cellviewID, then re-masters those instances with the master that matches the update library, update cell and update view names you provide. The * wildcard can be used in any search string to replace any number of characters prior to or following the asterisk. If any of the search strings are empty then all matches in the cds.lib are considered, if matches exist. If any of the update strings are empty the update string uses the equivalent search string value. If both the search and the update strings are empty, only exact matches of the respective strings are considered. Due to the extensive changes available when using this command, making a backup copy of your design to guard against errors when replacing data is recommended.

Arguments

d_cellViewId	The database ID of the cellview containing the instances you want to re-master.
t_searchLibrary	The library name to search for. Valid Values: any library name or empty string
t_searchCellview	The cell name to search for. Valid Values: any cell name or empty string
t_searchViewName	The view name to search for. Valid Values: any view name or empty string
t_updateLibrary	The library master used to re-master the search results. Valid Values: any library name or empty string
t_updateCellview	The cellview master used to re-master the search results. Valid Values: any cell name or empty string
t_updateViewName	The cellview master used to re-master the search results. Valid Values: any view name or empty string
g_checkTerminals	When `checkTerminals` is set to `t`, and the master instTerm names of the instances that match the search criteria is a matching set or subset of the instTerms of the destination cellview, the cellview will be re-mastered. If the instTerm names are not a matching set or subset, the cellviews will not be re-mastered. When `checkTerminals` is set to `t`, the following list shows the sample results, displaying the source, instTerm destination, and instTerm result. ■ ABC - ABC - re-mastered ■ ABC - ABCD - re-mastered ■ ABC - DEF - not re-mastered ■ ABC - AB - not re-mastered When `checkTerminals` is set to `nil`, the function does not consider the instTerm names, so the cellviews are re-mastered even if the instTerms are not a matching set or subset. Valid Values: `t` or `nil`

Value Returned

`t`	At least one of the searched instances is re-mastered.
`nil`	None of the searched instances met the update criteria and no instances are re-mastered.

Example

Remasters all instances of `designLib/fir_nib/abstract` in the specified cellview to `lefin_lib/fir_nib/layout`.

```
leRemasterInstances(cellViewId "designLib" "fir_nib" "abstract" "lefin_lib"
"fir_nib" "layout" nil)
```

leReplace

```
leReplace(
     d_cellViewId
     l_objects
     l_replaceValues
     )
     => t / nil
```

Description

Replaces the specified values on the specified list of objects in the cellview *d_cellViewId*.

Arguments

d_cellViewId	Database ID for the cellview where the objects are located.
l_objects	List of objects to be modified.
l_replaceValues	List of properties or attributes whose values are replaced. The list format is the same as *l_criteriaList* for leSearchHierarchy except the operator (==, >=, <=) is ignored.not re-mastered.

Value Returned

t	The values are replaced.
nil	The values are not replaced.

Example

Changes the layers of path1 and path2 to metal1 drawing and changes their widths to 5.0.

```
leReplace( cell2 list(path1 path2) list(list("layer" nil list("metal1" "drawing"))
list("path width" == 5.0)) )
```

leReplaceAnyInstMaster

```
leReplaceAnyInstMaster(
     d_instId
     { t_libName | nil }
     { t_cellName | nil }
     { t_viewName | nil }
     )
     =>t / nil
```

Description

Changes the master library, cell, or view name for the instance. If you do not want to change any of these names, you must type `nil` in its place.

Arguments

d_instId	The database ID of the instance.
t_libName	New master library name for the instance.
t_cellName	New master cell name for the instance.
t_viewName	New master view name for the instance.

Value Returned

`t`	The master library, cell, or view name for the instance is changed.
`nil`	The master library, cell, or view name for the instance is not changed.

Example

Changes the master cell name the instance to `Inv`.

```
foreach(fig geGetSelSet()
    if(fig~>objType == "inst"
        leReplaceAnyInstMaster(fig nil "Inv" nil)

        if(fig~>objType == "mosaic" && fig~>mosaicType == "simple"
            leReplaceAnyInstMaster(car(fig~>instanceList) nil "Inv" nil)
        )
    )
)
```

leSearchHierarchy

```
leSearchHierarchy(
     d_cellViewId
     l_bBox
     x_stopLevel
     t_objectType
     l_criteriaList
     )
     => l_objects / nil
```

Description

Searches the cellview *d_cellViewId* for objects that match the type and criteria specified. The search is bound by the specified *l_bBox* and restricted to the specified *x_stopLevel*. The search assumes all figures are valid if the Layer Selection Window (LSW) has not been initialized.

Arguments

d_cellViewId	Database ID for the cellview to search.
l_bBox	Bounding box within the cellview to search.
x_stopLevel	Specifies how deep in the hierarchy to search. Valid Values: any integer from `0` to `32`
t_objectType	Type of object for which to search. Valid Values: `inst`, `via`, `array`, `label`, `path`, `pathSeg`, `polygon`, `rectangle`, `ellipse`, `donut`, `trl`, `bend`, `taper`, `any shape`, `any conic`, `any microstrip`, `text display`, `pin`, `PR boundary`, `snap boundary`, `area boundary`, `cluster boundary`, or `blockage`.
l_criteriaList	List of criteria for which you want to search. You can search for one criterion or lists of several criteria. Valid Values: A list containing `t_criteriaType`, `t_operator`, and `g_value`: ■ `t_criteriaType` Criteria for which you want to search. Valid Values: any valid attribute associated with the specific object, or any additional property you created for the object. For details about any object attributes, see the description of the SKILL command used to create that object, for example, `leCreatePath`. ■ `t_operator` Boolean operator you want to use in the search. It must be appropriate for the criteria type. For example, <= is appropriate in a search for a numeric value, but not appropriate in a search for a string. Valid Values: `==`, `<`, `<=`, `>=`, `>`, `!=`, `EXIST` (property exists), `!EXIST` (property does not exist) ■ `g_value` Value of the criterion you want to find. Valid values include any value appropriate to this criteria type.

The following table lists the object types and the criteria that you can search for.

Object Type	Criteria Types
any conic	property, layer, any radius, net name
any microstrip	property, layer

Object Type	Criteria Types
any shape	property, layer, ROD name, net name
area boundary	property, boundary name
array	rows, columns, deltaX, deltaY, array name, lib name, cell name
bend	property, layer, bend style
blockage	property, layer name, blockage type, owner name
cluster boundary	property, boundary name, cluster name
donut	property, layer, inner radius, outer radius, any radius, net name
ellipse	property, layer, radiusX, radiusY, any radius, net name
instance	property, inst name, lib name, cell name, view name, mag, orient
label	property, layer, text, font, height, orient, justify, ROD name
path	property, layer, path style, path width, begin ext, end ext, ROD name, net name
pathSeg	property, layer, width, end style, begin style, net name
pin	property, layer, term name
polygon	property, layer, ROD name, net name
PR boundary	
rectangle	property, layer, ROD name, net name
snap boundary	
taper	property, layer, taper style
text display	property, layer, text, font, justify, height, orient, ROD name
trl [transmission line]	property, layer, bend style, chamfer fac, radius fac, trl width
via	property, via definition

Value Returned

l_objects	Returns *l_objects*, a list of all objects that meet the object type and search criteria.
nil	Does not return a list.

Examples

Searches all of cell2 at the top-level for paths on poly1 drawing layer with path width >= 3.0:

```
leSearchHierarchy( cell2 list( 0:0 300:300 ) 0 "path"
list( list( "layer" "==" list( "poly1" "drawing" ) )
list( "path width" ">=" 3.0 ) ) )
```

Searches for an exact cell with cell name myCell. The use of ^ and $ as used in the example limit the pattern matching. If the search string is myCell, without ^ and $, the function searches for all cells containing myCell in cell names:

```
cv=geGetWindowCellView()
leSearchHierarchy(cv cv~>bBox 0 "array" list(list("cell name" "==" "^myCell$")))
```

Searches all of the cellview bounding box 5 levels for all area boundaries in the cellview:

```
leSearchHierarchy(cv cv~>bBox 5 "area boundary" list())
```

Searches the cellview for an area boundary named B3:

```
leSearchHierarchy(cv cv~>bBox 5 "area boundary" list( list( "boundary name" "=="
"B3")))
```

Searches for all cluster boundaries in the cellview named myClusterBound which includes the cluster name myCluster containing the property prop1 having a value of val1:

```
criteriaList1 = list( list("boundary name" "==" "myClusterBound")
                      list( "property" "==" list("prop1" "val1"))
                      list( "cluster name" "==" "myCluster")
                    )
leSearchHierarchy(cv cv~>bBox 5 "cluster boundary" criteriaList1)
```

Searches for all blockages in the cellview:

```
leSearchHierarchy(cv cv~>bBox 5 "blockage" list())
```

Searches for all the routing blockages in the cellview on the metal1 layer:

```
criteriaList2 = list( list("blockage type" "==" "routing")
                      list("layer name" "==" "metal1")
                    )
leSearchHierarchy(cv cv~>bBox 5 "blockage" criteriaList2)
```

Searches for all path segments in the cellview with a begin and end style of extend and with a property of `prop2` having a value of `val2`:

```
criteriaList = list( list("begin style" "==" "extend")
                     list("end style" "==" "extend")
                     list("property" "==" list("prop2" "val2"))
                   )
leSearchHierarchy(cv cv~>bBox 5 "pathSeg" criteriaList)
```

Interactive Function

Enter this function with only the window ID argument; the system prompts you to specify the object type, criteria type, and where to search. If you do not specify *`w_windowId`*, the current window is used.

```
leHiSearch( [ w_windowId ] ) => t / nil
```

Hierarchy Traversal Functions

You can use SKILL functions to traverse the design hierarchy to edit the master of an instance. For example, you can use `leEditInPlace` to descend through a hierarchy to edit a cell in place.

You can also use the SKILL function related to embedded module hierarchy to access the Uniquify form.

Related Topics

leDescend

leDoubleClick

leEditInPlace

leEIPZoomAbsoluteScale

leUniquifyCellView

leDescend

```
leDescend(
     w_windowId
     d_instId
     [ x_row ]
     [ x_column ]
     )
     => w_windowId / nil
```

Description

Descends through the hierarchy to open the master of instance *d_instId*. leDescend attempts to open the cellview in the same mode as the current cellview, but if an editable version is not available, leDescend opens the master in read-only mode.

Arguments

w_windowId	Window ID of the window containing the instance.
d_instId	The database ID of the instance to descend into.
x_row	Row position if the instance is part of a mosaic.
x_column	Column position if the instance is part of a mosaic.

Value Returned

w_windowId	Returns the window ID of the window containing the instance if the master instance cellview opens. (The value for *w_windowId* is the same value provided in the argument *w_windowId* if leDescend is successful.)
nil	The master instance cellview does not open.

Example

Descends into the master instance of myInst using the current Virtuoso Layout Suite editing window. The returned value is the window ID.

```
leDescend(hiGetCurrentWindow() myInst)
```

leDoubleClick

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

Description

Runs the Edit In Place command on the object if the object at the point of double-click is either an instance or a group.

Arguments

None

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

Example

EIPs into the instance or group at the point of double-click.

```
leDoubleClick()
```

leEditInPlace

```
leEditInPlace(
     w_windowId
     l_hierlist
     )
     => t / nil
```

Description

Descends through the hierarchy to edit a cell in place.

Arguments

w_windowId	Window ID of the window to use.
l_hierlist	List describing a path for the instance to edit in place. This argument takes the form `list((`*instId memInst row col*`) (`*instId memInst row col*`) ...)` The *row* and *col* elements represent the row and column positions of the instance if it is part of a mosaic. *memInst* is an integer that is not used in `leEditInPlace`.

Value Returned

`t`	The cell is made editable in the specified window.
`nil`	The cell is not made editable in the specified window.

Example

Descends through an instance *topInst* at level one, an element of a mosaic *Sub1Inst* at level two, and an instance *sub2Inst* at level three. Here, *topInst*, *sub1Inst*, and *sub2Inst* are the appropriate database IDs stored in variables.

```
leEditInPlace( hiGetCurrentWindow() list(list(topInst 0 0 0) list(sub1Inst 0 1 2)
list(sub2Inst 0 0 0)) )
```

leEIPZoomAbsoluteScale

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

Description

Fits the master cell of the instance you want to edit in the window. This function also supports edit in place for fig groups. The window is redrawn so the edit-in-place cell or fig group fills it.

Arguments

None

Value Returned

t	The command executed successfully.
nil	The command did not execute successfully.

leUniquifyCellView

```
leUniquifyCellView(
     )
```

Description

Opens the *Uniquify* form to specify the library, cell, and view name of the cellview to be uniquified. Alternatively, you can use the *Uniquify* command on the *Tools* menu in the CIW. A message displays after the successful uniquification of the cellview. If the cellview is already uniquified or the cellview cannot be opened, the command exits and displays an appropriate message.

Arguments

None

Value Returned

None

Binary and Unary Functions

The binary and unary SKILL functions enable you to create objects from the union, intersection, or non-intersection of existing objects. For example, you can create shapes in `cell1` on layer `activegate` from the intersection of all shapes on the `poly1` and `pdiff` layers.

Related Topics

leLayerAnd

leLayerAndNot

leLayerOr

leLayerSize

leLayerXor

leLayerAnd

```
leLayerAnd(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes / nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that correspond to the intersections of all shapes on layers *g_lpp1* and *g_lpp2*. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	The database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes	The IDs of the new shapes if new shapes are created.
nil	New shapes are not created.

Example

Creates shapes in `cell1` on layer `activegate` from the intersection of all shapes on the `poly1` and `pdiff` layers. Returns the shape IDs of the shapes.

```
leLayerAnd( cell1 list("poly1" "drawing") list("pdiff" "drawing")
list("activegate" "drawing") )
```

leLayerAndNot

```
leLayerAndNot(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes / nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that correspond to the shapes on layer *g_lpp1* minus the intersections with the shapes on layer *g_lpp2*. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	The database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
*g_lpp*3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes	Returns the new shapes if they are created.
nil	The new shapes are not created.

Example

Creates shapes in `cell3` on the layer `opencontacts` from shapes on the layer `via` minus the intersections with shapes on layer `metal2`. Returns the shape IDs of the shapes.

```
leLayerAndNot( cell3 list("via" "drawing") list("metal2" "drawing")
list("opencontacts" "drawing") )
```

leLayerOr

```
leLayerOr(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes / nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that correspond to the union of shapes on layers *g_lpp1* and *g_lpp2*. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	The database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes	Returns the IDs of the new shapes if they are created.
nil	The new shapes are not created.

Example

Creates shapes in `cell3` on the layer `allmetal` from the union of the shapes on the `metal1` and `metal2` layers. Returns the shape IDs of the shapes.

```
leLayerOr( (cell3 list("metal1" "drawing") list("metal2" "drawing")
list("allmetal" "drawing") )
```

leLayerSize

```
leLayerSize(
     d_cellViewId
     g_lpp1
     n_sizeAmount
     g_lpp2
     )
     => l_shapes / nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp2* by over sizing the shapes on layer *g_lpp1* by *n_sizeAmount* number of units, which can be either positive (oversize) or negative (undersize). A layer-purpose pair is either a layer name or a list containing a layer name followed by a layer purpose.

Arguments

d_cellViewId	The database ID of the cellview containing the layer-purpose pairs.
g_lpp1	Layer from which the selected shapes are sized.
n_sizeAmount	Number of user units used to oversize or undersize the shapes.
g_lpp2	Layer on which new shapes are created.

Value Returned

l_shapes	The new shapes if they are created.
nil	The new shapes are not created.

Example

Creates shapes in `cell2` on layer `implant` by over sizing the shapes on layer `gate` by 1.5 user units. Returns the shape IDs of the shapes.

```
leLayerSize( cell2 list("gate" "drawing") 1.5 list("implant" "drawing") )
```

leLayerXor

```
leLayerXor(
     d_cellViewId
     g_lpp1
     g_lpp2
     g_lpp3
     )
     => l_shapes / nil
```

Description

Creates shapes in cellview *d_cellViewId* on layer *g_lpp3* that are the result of the union of the shapes on layers *g_lpp1* and *g_lpp2* minus their intersection. A layer-purpose pair is a list containing a layer name followed by a layer purpose.

Arguments

d_cellviewId	The database ID of the cellview containing the layer-purpose pairs.
g_lpp1	First layer-purpose pair.
g_lpp2	Second layer-purpose pair.
g_lpp3	Layer-purpose pair assigned to the new shapes created.

Value Returned

l_shapes	Returns the new shapes if they are created.
`nil`	The new shapes are not created.

Example

Creates shapes in `cell2` on the layer `missingwell`. The new shapes are the result of the union of the shapes on the `pdiff` or `nwell` layers minus their intersections. Returns the shape IDs of the shapes.

```
leLayerXor( cell2 list("pdiff" "drawing") list("nwell" "drawing")
list("missingwell" "drawing") )
```

Bindkey Functions

You can use bindkey SKILL functions to customize the bindkeys in Virtuoso Layout Suite. For example, you can choose to implement a command option for create and edit commands when a bindkey is pressed individually or in combination with `Ctrl` or `Shift` key.

Related Topics

cmdCtrlOption

cmdOption

cmdShiftOption

leArrowFunc

leSelBoxOrStretch

leSpaceBarFunc

Bindkeys and Access Keys

cmdCtrlOption

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

Description

Implements a command option for Virtuoso Layout Suite XLcreating and editing commands when you press the `Control` key and press the bindkey. The command options are embedded in the Virtuoso Layout Suite XL software and cannot be changed. The default bindkey for this function is the right mouse button. You can change the default to another key.

Create Path Command — While creating a path, press `Control` and click the bindkey to change the layer on which the path is being created. You must have visible, valid, and selectable layers set and you must have selected a valid entry level for a path.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Sets the *Change Path Layer* command option for the *Create Path* command to the `a` key.

```
hiSetBindKey("Layout" "Ctrl<Key>a EF" "cmdCtrlOption()")
```

Related Topics

Bindkeys and Access Keys

cmdOption

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

Description

Implements a command option for Virtuoso Layout Suite XL creating and editing commands when you press the bindkey. The command options are embedded in the Virtuoso Layout Suite XL software and cannot be changed. The default bindkey for this function is the right mouse button. You can change the default to another key.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Sets the command options for Virtuoso Layout Suite XL creating and editing commands to the `a` key.

```
hiSetBindKey("Layout" "Ctrl<Key>a EF" "cmdOption()")
```

Additional Information

The following are the supported commands and command options:

- Chop — When you are in polygon or line mode of the *Chop* command, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.
- Create Path — When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

- Create Pin — When creating a polygonal shape pin, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.
- Create Polygon — When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.
- Reshape — When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*. When you have finished entering the points, the bindkey toggles between the two shapes you can select.
- Split — When you are entering points, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*. When you have finished entering the points, the bindkey toggles between the two shapes you can select.
- Trl — When you are entering points of a transmission line, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.
- Yank — When you are in polygon mode, you can toggle the snap mode between *L90XFirst* and *L90YFirst* after you have set the snap mode to *L90XFirst* or *L90YFirst*.

Related Topics

Bindkeys and Access Keys

cmdShiftOption

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

Description

Implements a command option for Virtuoso Layout Suite L creating and editing commands when you press the `Shift` key and press the bindkey. The command options are embedded in the Virtuoso Layout Suite L software and cannot be changed. The default bindkey for this function is the right mouse button. You can change the default to another key.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Sets the command options for Virtuoso Layout Suite L creating and editing commands to the `a` key.

```
hiSetBindKey("Layout" "Shift<Key>a EF" "cmdShiftOption()")
```

Additional Information

The following are the supported commands and command options:

- Copy — When you are copying an item or items, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).
- Move — When you are moving an item or items, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).
- Create Instance — When you are creating an instance, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

- Create Label — When you are creating a label, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis). The letters will display right side up and will not display backwards.
- Create Contact — When you are creating polygonal shaped contacts or multiple rows and columns of contacts, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).
- Paste — When you are pasting an item or items, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).
- Generate Selected From Source — When you are using the *Generate Selected From Source* command, you can toggle between *MX* (mirrored over the X axis) and *MY* (mirrored over the Y axis).

leArrowFunc

```
leArrowFunc(
     t_keyName
     [ g_optionalArg ]
     )
     => return value from the called function or function bound by default or nil
```

Description

Executes the function specified in the layout environmental variable `spaceFuncName`, passing the ID of the current window, the key name, and the optional argument. If `spaceFuncName` is empty, this function executes the current non-enter function binding for the specified key name.

Arguments

t_keyName	Text string specifying key name. Do not include <Key> in the argument passed to the function.
g_optionalArg	An optional argument to pass to the function.

Value Returned

Passes on the return value from the called function, or the function bound by default, or `nil`.

Example

```
hiSetBindKeys("Layout"
   list(list("<Key>Up EF" "leArrowFunc('Up')")
     list("Ctrl<Key>Up EF" "leArrowFunc('Up' t)")
     list("<Key>Down EF" "leArrowFunc('Down')")
     list("Ctrl<Key>Down EF" "leArrowFunc('Down' t)")))
leHiCreateWire()
;; ...
```

where `('Up')` represents pressing the Up arrow key and `('Up' t)` represents pressing the Control key with the Up arrow key. When you are creating a wire, this sample code lets you press the Up or Down arrow key to place a default via to the next layer, above or below, respectively.

leSelBoxOrStretch

```
leSelBoxOrStretch(
     [ w_windowId ]
     )
     => t
```

Description

The Virtuoso Layout Suite L binds this function to the mouse button DrawThru1 action. The default behavior for mouse button DrawThru1 is select by bBox. When you select using a bBox, rather than a point, the software checks what is under the selected area. If something under the initial point of the bBox is already selected, the software automatically calls the stretch command. If nothing under the cursor is selected, the software selects the bBox you have drawn. If *w_windowId* is not specified, the current window is used. This function must not be changed.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The command executed successfully.

leSpaceBarFunc

```
leSpaceBarFunc(
     [ w_windowId ]
     [ g_useDefault ]
     [ g_shift ]
     )
     => return value from the called function
```

Description

Invokes the function specified by name in the Virtuoso Layout Suite L environment variable `spaceFuncName` when the *`g_useDefault`* argument is set to `t`. When the *`g_useDefault`* argument is set to `nil` the space bar is not used as a bindkey. If *`w_windowId`* is not specified, the current window is used.

Arguments

`w_windowId`	Window ID of the window specified.
`g_useDefault`	If a default via can be determined, it will be selected. Valid Values: `t` or `nil`
`g_shift`	Space bar can be used in conjunction with the shift key. Valid Values: `t` or `nil`

Value Returned

`return value from the called function`

Return value from the called function.

Example

```
\a leHiCreateWire()
\a mouseAddPoint()
\i 10.15:19.34 ;user initiates new path or pathSeg here
\a leSpaceBarFunc(hiGetCurrentWindow() t
)
```

This command will be generated in the log during execution of commands such as *Create – Wire* where the space bar can be used as a convenient way to add a via. In this example the user selects a via to add, the default if available. The *Add Via* form will appear if a default via is not available.

Menu Builder Functions

You can use menu builder SKILL functions to perform tasks such as create an action and register it with the menu builder for future retrieval and use within menus and retrieve an action from the menu builder data structure.

Related Topics

hiMakeLPChoiceList

mbGetAction

mbRegisterAction

mbRegisterCustomMenu

mbRegisterHierMenu

mbRegisterMenuItem

mbSetContextData

mbUnregisterAction

hiMakeLPChoiceList

```
hiMakeLPChoiceList(
     d_techFileID
     l_layerName&Purpose
     [ g_layerNum ]
     [ g_swatchIcon ]
     )
     => l_iconList / nil
```

Description

Creates a layer purpose icon list for use in a form field or menu.

Arguments

d_techFileID	Technology File ID in which the layers are defined.
l_layerName&Purpose	List of lists identifying layers and their purposes.
g_layerNum	Layer number
g_swatchIcon	A boolean value specifying whether or not the icon will contain only the layer swatches. No layer description will be available in the new icon if this argument is set to `t`.

Value Returned

l_iconList	Returns *l_icon list* if the layer purpose icon list was created.
`nil`	Returns `nil` if the layer purpose icon list was not created.

Example

Creates a layer purpose icon list with two layers, a metal1 drawing layer and a metal2 drawing layer:

```
iconList = hiMakeLPChoiceList(techFileId list(
    list("metal1" "drawing")
    list("metal2" "drawing")
    )
)
```

Creates a layer purpose icon list containing all the layers of the specified technology file:

```
iconList = hiMakeLPChoiceList(techFileId list())
```

Returns a list containing the icon along with layer and purpose name.The layer and purpose name can be extracted from this list and passed to `hiCreateMenuItem` as `itemText`:

```
l_choices=(hiMakeLPChoiceList (techGetTechFile (getWindowRep)) '(
("y0" "drawing")("y1" "drawing")("y2" "drawing")
)
nil t)
```

mbGetAction

```
mbGetAction(
     t_viewType
     t_uniqueId
     )
     => o_action / nil
```

Description

Retrieves an action from the menu builder data structure.

Arguments

t_viewType	Name of the view type for which this action has been registered.
t_uniqueId	Unique ID for the reference action.

Values Returned

o_action	ID of the `hi` action that is retrieved.
`nil`	No `hi` action is found with the specified arguments.

Example

Retrieves the action `del` from menu builder data structure for the `maskLayout` view type.

```
action=mbGetAction("maskLayout" "del")
```

Returns `del`.

mbRegisterAction

```
mbRegisterAction(
     t_viewType
     t_uniqueId
     t_name
     t_callback
     t_enableCB
     [ ?iconFile g_iconFile ]
     [ ?icon g_icon ]
     [ ?checkable g_checkable ]
     [ ?checkedCB t_checkedCB ]
     )
     => o_action / nil
```

Description

Creates an action and registers it with the menu builder for future retrieval and use within menus. This action will appear only when mbSetContextData is used to set its visibility location.

Arguments

t_viewType	Name of the view type for which this action has been registered.
t_uniqueId	Unique ID for the reference action.
t_name	Name of the registered action.
t_callback	Function called when the action is selected.
t_enableCB	Function to determine if the action is enabled or disabled. This function is called each time a menu is mapped, and allows a decision whether the menu will gray out or not at the time of menu mapping. Function returns `t` if the action is enabled and returns `nil` if the action is disabled.
`?iconFile` *g_iconFile*	Name of the icon associated with the action.
`?icon` *g_icon*	Icon associated with the action. The function reads the icon file from the `share/cds/icons` directory in the installation path.
`?checkable` *g_checkable*	Allows the action to appear with a check box.
`?checkedCB` *t_checkedCB*	Function called each time the status of the check box is changed.

Values Returned

o_action	ID of the action that is created and registered.
`nil`	Returned when no action is created and its registration fails.

Example

```
mbRegisterAction("maskLayout" "del" "Delete" "leHiDelete()" "geGetSelSet() && t"
?icon "delete.png")
mbSetContextData("maskLayout" "del" "Shape Instance" "Navigator Canvas" "Common")
```

Returns `del`.

The menu will appear only if you have something selected on the canvas or in the navigator.

mbRegisterCustomMenu

```
mbRegisterCustomMenu(
     t_viewType
     t_uniqueId
     t_enableCB
     t_customCB
     )
     => t / nil
```

Description

Registers a custom menu with the menu builder for future retrieval and use within menus. A custom menu item is a single or slider menu that is built using the `hi` function calls, independently of the menu builder. The `t_customCB` argument is used to specify the function that is responsible for building and returning the menu item. The menu will display only when the mbSetContextData function is called.

Arguments

t_viewType	Name of the view type for which this action has been registered.
t_uniqueId	Unique ID for the reference custom menu.
t_enableCB	Function to determine if action is enabled or disabled.
t_customCB	Function to return the menu item or the slider menu item.

Values Returned

`t`	Returned when the custom menu is successfully registered.
`nil`	Returned when registration of the custom menu fails.

Example

Here, the first function, `BuildCustMenuItem()` creates a pulldown menu, and returns a slider menu containing it. The second function `EnableCustMenuItem()` returns `t` to ensure that the menu item is always enabled .

```
procedure(BuildCustMenuItem()
    hiCreatePulldownMenu(
      'MyCustomMenu
```

```
      "Custom Menu"
      list(
    hiCreateMenuItem(
      ?name 'item1
      ?itemText "Item 1"
      ?callback "println(\"Item 1\")"
    )
    hiCreateMenuItem(
      ?name 'item2
      ?itemText "Item 2"
      ?callback "println(\"Item 2\")"
    )
      )
    )
    hiCreateSliderMenuItem(
      ?name 'MySlider
      ?itemText "Custom Menu"
      ?subMenu MyCustomMenu
    )
  )
  procedure(EnableCustMenuItem()
    t
  )
  mbRegisterCustomMenu("maskLayout" "custMenu" "EnableCustMenuItem()"
"BuildCustMenuItem()")
  mbSetContextData("maskLayout" "custMenu" "Via Ruler" "Canvas" "Common")
```

Returns `t`.

mbRegisterHierMenu

```
mbRegisterHierMenu(
     t_viewType
     t_uniqueId
     t_name
     t_enableCB
     )
     => t / nil
```

Description

Registers a hierarchical menu with the menu builder for future retrieval and use within menus.

Arguments

t_viewType	Name of the view type with which the menu is to be registered.
t_uniqueId	Unique ID for the reference menu.
t_name	Name of the registered menu.
t_enableCB	Function to determine if action is enabled or disabled.

Values Returned

`t`	Returned when the menu is successfully registered.
`nil`	Returned when registration of the menu fails.

Example

Creates a slider (hierarchical) menu called `Hier Example` under the `Shape` context menu, with two sub items, labeled `Item 1` and `Item 2`.

```
mbRegisterAction("maskLayout" "subItem1" "Item 1" "println(\"Item 1\")" "t")
mbRegisterAction("maskLayout" "subItem2" "Item 2" "println(\"Item 2\")" "t")
mbRegisterHierMenu("maskLayout" "hierMenu" "Hier Example" "t")
mbSetContextData("maskLayout" "subItem1" "Shape" "Canvas" "Common" ?parent
"hierMenu")
mbSetContextData("maskLayout" "subItem2" "Shape" "Canvas" "Common" ?parent
"hierMenu")
mbSetContextData("maskLayout" "hierMenu" "Shape" "Canvas" "Common")
mbRegisterMenuItem
```

mbRegisterMenuItem

```
mbRegisterMenuItem(
     t_viewType
     l_menuItemDef
     t_enableCB
     )
     => t_menuItemId / nil
```

Description

Registers a menu item with the menu builder that has been defined using the older menu creation syntax. mbSetContextData needs to be called before the menu item is visible.

Arguments

t_viewType	Name of the view type for which this action has been registered.
l_menuItemDef	List containing the old style menu definition. The list is specified in the following format: `<name> <menuText> <callBack>`
t_enableCB	Function to determine if action is enabled or disabled.

Values Returned

t_menuItemId	ID of the menu item that is created and registered.
`nil`	Returned when registration of the menu fails.

Example

```
MyNewItem = '(NewItem "&New..." "deFileNew()")
mbRegisterMenuItem("maskLayout" MyNewItem "t")
mbSetContextData("maskLayout" "NewItem" "Shape" "Canvas" "Common")
```

Returns `hiMenuItem@0x1cdaa020, t`.

Note: You can use `symbolToString(car(MyNewItem))` to get the second argument for the function `mbSetContextData()`.

Registers the menu item `MyNewItem` with the menu builder. mbSetContextData is called to make this menu item visible. Also, the unique ID of the menu item, `NewItem` is the first entry in the sublist passed as the second argument to `mbRegisterMenuItem`.

mbSetContextData

```
mbSetContextData(
     t_viewType
     t_uniqueId
     t_validObjs
     t_validWidgets
     t_grouping
     [ ?parent t_parent ]
     [ ?removeWhenDisabled g_removeWhenDisabled ]
     )
     => t / nil
```

Description

Sets various attributes to determine the menus with which the registered action is to be associated.

Arguments

t_viewType	Name of the view type for which this action has been registered. Valid Values: Schematic: `schematic`, `schematicXL`, `analogArtist-schematic`, `adexl-schematic`, `adegxl-schematic`, `hman-schematic`, `amsArtist-schematic`, `mixedSignalArtist-schematic`, `ncVlogSchematic`, `spectre-schematic`, `other-schematic`, `UltraSim-schematic`, `VHDLToolbox`, `verilog`, `schematicInteractive`, `mspsSchematicApplication` Layout: `maskLayout`, `maskLayoutParamCell`, `maskLayoutXL`, `maskLayoutGXL`, `maskLayoutCE`, `analogArtist-maskLayout`, `adexl-maskLayout`, `adegxl-maskLayout`, `other-maskLayout`, `spectre-maskLayout`, `UltraSim-maskLayout`, `maskLayoutIQView`, `parasitics-MaskLayout`, `verilogMaskLayout`, `maskLayoutVSA`
t_uniqueId	Unique ID to reference the action
t_validObjs	Space separated list of objects for which the action is valid Valid Values: `Instance, Net, Shape, Pin, Via, Group, Clone, Modgen, Ruler, Marker, FGuardRing, None, Pcell, Ungenerated, Boundary, Blockage, Row, Mosaic, ModInst, RowRegion, Any` Here you specify: `Any` , if the menu item is needed for all objects. `Ungenerated`, if an instance or pin is in the schematic, but not currently in the layout. These items are selectable only in the navigator and so it is the only place where you can access the menu for them. `Modgen`, for the group of objects created with the Module Generator tool. `None`, to display the item when nothing is selected.

t_validWidgets	Space separated list of widgets for which the action is valid Valid Values: `Canvas` and `Navigator`
t_grouping	Name of the group of menu items in which the registered action is to appear. Valid Values: `Create`, `Edit`, `Hierarchy`, `Groups`, `ObjSpecific`, `Secondary`, `Common`, `Constraints`, and `Properties`
`?parent` *t_parent*	Name of the parent menu if this action is to be part of a hierarchical menu. The parent menu should already be registered.
`?removeWhenDisabled` *g_removeWhenDisabled*	When `t`, this argument removes the action instead of graying it out when disabled.

Values Returned

`t`	Returned when the context data is successfully set.
`nil`	Returned when setting of the context data fails.

Example

Adds a new function to the via shortcut menu:

```
mbRegisterAction("maskLayout" "uniqueName"  "Pop-up menu text"
"CustomerFunction()" "t")
mbSetContextData("maskLayout" "uniqueName"  "Via"  "Navigator Canvas"
"ObjSpecific")
```

If the menu item is valid for all objects replace `Via` in `mbSetContextData` with `Any`.

mbUnregisterAction

```
mbUnregisterAction(
     t_name
     [ ?objTypes t_objTypes ]
     [ ?viewType t_viewType ]
     [ ?silent g_silent ]
     )
     => t / nil
```

Description

Removes the provided menu item from the RMB menu. The optional arguments can be used to limit the removal of a menu item to specific object types or view types for which it is registered.

Arguments

t_name Name of the action which is to be removed. The name is the text that appears in the context menu.

?objTypes *t_objTypes*

Name of the object. Can be provided to limit the removal to a specific type of object, For example, "Instance". Default is "All"

?viewType *t_viewType*

Name of the view type. Can be provided to specify which viewType the action was registered with. For example. "maskLayout". The default is nil, in which case all viewTypes will be looked at for a matching action.

?silent *g_silent*

This is a boolean argument. Can be provided to suppress the information messages which are output stating the actions that have been removed. The default is nil.

Values Returned

t	Returned when the menu item has been succesfully removed.
nil	Returned when the menu item has not been succesfully removed.

Example

Removes the *Rotate Left* command from all shortcut menus:

```
\i mbUnregisterAction("Rotate Left")
\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Blockage' context menu.
\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Boundary' context menu.
\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Clone' context menu.
\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Group' context menu.
\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Instance' context menu.
```

```
\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Modgen' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Mosaic' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'None' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Pin' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Row' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Ruler' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Shape' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Via' context menu.

\t t

\p >
```

Removes the *Rotate Left* action and limit by object type:

```
\i mbUnregisterAction("Rotate Left" ?objTypes "Instance Mosaic Group")

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Group' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Instance' context menu.

\o INFO (MB-1002): The 'Rotate Left' menu item has been removed from the
'maskLayout' 'Mosaic' context menu.

\t t

\p >
```

Interactive Functions

Interactive SKILL functions help you use forms and menu commands. These interactive functions are associated with layout editor procedural functions. For example, the `hiLayerDispMainForm` opens the Layer Purpose Pair Editor form while the `leCloseWindow` closes the current layout window.

Related Topics

hiLayerDispMainForm

leCloseWindow

leDesignSummary

leEditDesignProperties

hiLayerDispMainForm

```
hiLayerDispMainForm(
     )
     => t
```

Description

Opens the Layer Purpose Pair Editor form. You can also access this form using the *CIW – Tools – Technology File Manager – Edit Layers* command.

Arguments

None

Value Returned

`t`	The function is successful.

leCloseWindow

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

Description

Closes the current layout window. Similar to the `hiCloseWindow` function. To customize a bindkey you must use `leCloseWindow` because it is the function used in the callback of the *File – Close* menu command.

Arguments

None

Value Returned

`t`	Returns `t` if the function is successful.
`nil`	Returns `nil` if the function is not successful.

leDesignSummary

```
leDesignSummary(
     d_cellViewId
     t_fileName
     g_append
     [ w_windowId ]
     [ basicSummary ]
     [ netStats ]
     [ connectivityStats ]
     [ routingStats ]
     )
     => t / nil
```

Description

Stores or outputs leHiSummary results to an external specified file.

Arguments

d_cellViewId	Any cellview id. If not specified, the current cellview id is used.
t_fileName	Name of the output file.
g_append	Appends to existing file if `t`, overwrites output file if `nil`. Valid Values: `t` or `nil`
w_windowId	Window ID of the window specified. If *w_windowId* is specified, the display levels (Start Level and Stop Level) are printed to the specified file. If *w_windowId* is not specified, the display levels are not printed to the file.
basicSummary	Prints the summary information to the specified file. Valid Values: `t` or `nil`
netStats	Prints the nets statistics information to the specified file. Valid Values: `t` or `nil`
connectivityStats	Prints the connectivity statistics information to the specified file. Valid Values: `t` or `nil`
routingStats	Prints the routing statistics information to the specified file. Valid Values: `t` or `nil`

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Examples

Appends the cellview summary to `filename` file:

```
leDesignSummary(cellViewId "fileName" t)
```

Overwrites the cellview summary to `filename` file:

```
leDesignSummary(cellViewId "fileName" nil)
```

Appends the cellview summary from the specified window to `filename` file:

```
leDesignSummary(cellViewId "fileName" t w_windowId)
```

Overwrites the cellview summary from the specified window to `filename` file.:

```
leDesignSummary(cellViewId "fileName" nil w_windowId)
```

Appends the cellview summary, basic summary, net statistics, connectivity statistics, and routing statistics from the specified window to filename file:

```
leDesignSummary(cellViewId "fileName" t w_windowId t t t t)
```

leEditDesignProperties

```
leEditDesignProperties(
     [ d_cellviewId ]
     )
     => t / nil
```

Description

Opens the Edit Cellview Properties form, which lets you change the design properties in the cellview *d_cellViewId*. If *d_cellViewId* is not specified, the cellview in the current window is used.

You can view the properties and values listed under *Attribute*, but you cannot change them. You can add, modify, delete or change the properties and values listed under *Property*.

Arguments

d_cellViewId	Any cellview id. If not specified, the current cellview id is used.

Value Returned

t	Returns t if the function is successful.
nil	Returns nil if the function is not successful.

leExportLabel

```
leExportLabel(
     [ t_fileName ]
     )
     => t / nil
```

Description

Saves the label information in the current cellview to the specified file.

Arguments

t_fileName	Name of the file to which label information is written. If the file already exists, the system prompts you to overwrite it.

Value Returned

`t`	Label information was saved to the specified file.
`nil`	Label information was not saved to the specified file. The system issues a message to let you know why the operation failed.

Examples

Saves the label information in the current cellview to a file called `labelInfoFile`.

```
leExportLabel("labelInfoFile")
```

leGetCoordinateForm

```
leGetCoordinatesForm(
     )
```

Description

Opens the Enter Points form which is used to specify the coordinates required to allow a command to proceed.

Note: When used without an already running command, the `leGetCoordinatesForm()` function works like the pan to coordinate function.

Arguments

None

Value Returned

None

Example

Opens the Enter Points form.

```
leGetCoordinateForm()
```

leHiAbout

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

Description

Opens the product information window, which includes the release number and copyright information.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiAddShapeToNet

```
leHiAddShapeToNet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Add Shape To Net form, which lets you associate a shape with a net. If *w_windowId* is not specified, the current window is used.

You can add selected shapes to a specified net or based on the pins overlapping the selected shapes.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiAddToGroup

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

Description

Opens the Select Target Group form to add shapes to an existing group.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiAlign

```
leHiAlign(
     )
     => t
```

Description

Opens the Align form, and returns the value of `t`.

Arguments

None

Value Returned

`t`	The function is successful.

leHiAttach

```
leHiAttach(
     [ w_windowId ]
     )
     => t / nil
```

Description

Attaches a selected object in the specified window to another object in the window. The attached object is called the *child* object, and the object it is attached to is called the *parent* object. When you move, copy, or delete the parent, the child is also moved, copied, or deleted. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCellviewTrackPatterns

```
leHiCellviewTrackPatterns(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Track Pattern Editor form, which lets you add, update or delete track patterns. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiChop

```
leHiChop(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Chop form, which lets you remove parts of objects or break objects into multiple objects. If *w_windowId* is not specified, the current window is used.

You are prompted to create a shape in the cellview that the system uses as a cutter to determine where to remove part of the objects or where to break them into multiple objects.

Chop cannot be used on pins. Use *Reshape* or *Stretch* to change pin shapes.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiClearRuler

```
leHiClearRuler(
     [ w_windowId ]
     )
     => t / nil
```

Description

Clears the rulers that stay in the window when the *Keep Ruler* option is selected using `leHiCreateRuler`. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiConvertInstToMosaic

```
leHiConvertInstToMosaic(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Convert To Mosaic* command, which lets you convert one or more selected instances either into an equivalent number of mosaics or into a single mosaic. If *w_windowId* is not specified, the current window is used. You are prompted to select one or more instances to convert into mosaics.

To convert instances to mosaics procedurally, see leConvertInstToMosaic.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiConvertPolygonToPath

```
leHiConvertPolygonToPath(
     [ w_windowId ]
     )
     => t / nil
```

Description

Converts a selected polygon in the specified window to a path if a uniform width path can be inferred. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	ID of the window in which polygon needs to be converted to a path.

Values Returned

t	Returned when the selected polygon has been converted to a path.
nil	Returned when the selected polygon has not been converted to a path.

Examples

Converts the selected polygon to path in the current window.

```
leHiConvertPolygonToPath()
```

Related Topics

leConvertPolygonToPath

leHiConvertShapeToPathSeg

```
leHiConvertShapeToPathSeg(
     [ w_windowId ]
     )
     => t / nil
```

Description

Converts a selected polygon, rectangle, or path in the specified window to a pathSeg, if a uniform width path can be inferred. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	ID of the window in which the polygon, rectangle, or path needs to be converted to a pathSeg.

Values Returned

t	Returned when the selected polygon, rectangle, or path has been converted to a pathSeg.
nil	Returned when the selected polygon, rectangle, or path has not been converted to a pathSeg.

Examples

Converts the selected polygon, rectangle, or path to pathSeg in the current window.

```
leHiConvertShapeToPathSeg()
```

leHiConvertShapeToPolygon

```
leHiConvertShapeToPolygon(
     [ w_windowId ]
     )
     => t / nil
```

Description

Converts a selected object in the specified window to a polygon. Converts rectangles, paths, donuts, circles, and ellipses using the number of conic sides specified using `leHiEditEditorOptions`. If *`w_windowId`* is not specified, the current window is used.

Arguments

`w_windowId`	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCopy

```
leHiCopy(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Copy form, which lets you copy selected objects in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to point to a reference point for the copied objects. You can change the layer, rotation, and snap mode when you copy the object. You can also create an array of the copied objects.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateAreaBoundary

```
leHiCreateAreaBoundary(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Area Boundary form, which lets you create an Area Boundary. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateBend

```
leHiCreateBend(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Transmission Line Bend form, which lets you create a transmission line bend in the specified window. If *w_windowId* is not specified, the current window is used.

You can set the bend style (bend, chamfer, or radial), the factors for the bend style, and the resolution for the bend. You can also set the snap mode.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateBlockage

```
leHiCreateBlockage(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Blockage form, which lets you create blockages. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateChoiceOfPin

```
leHiCreateChoiceOfPin(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Pin form in *w_windowId*. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateCircle

```
leHiCreateCircle(
     [ w_windowId ]
     )
     => t / nil
```

Description

Creates a circle in the specified window. You are prompted to point to a location for the center of the circle and the outer edge of the circle. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateClusterBoundary

```
leHiCreateClusterBoundary(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Cluster Boundary form, which lets you create a cluster boundary. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateClusters

```
leHiCreateClusters(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the *Create – P&R Objects – Clusters* form, which lets you create and edit clusters and sub-clusters. If `w_windowId` is not specified, the current window is used.

Arguments

`w_windowId`	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCreateDonut

```
leHiCreateDonut(
     [ w_windowId ]
     )
     => t / nil
```

Description

Creates a donut in the specified window. You are prompted to point to a location for the center of the donut, the inner edge of the donut, and the outer edge of the donut. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateEllipse

```
leHiCreateEllipse(
     [ w_windowId ]
     )
     => t / nil
```

Description

Creates an ellipse in the specified window. You are prompted to point to the corners of the bounding box for the ellipse. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateGroup

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

Description

Opens the Create Group form to select figures to form a new group.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCreateGuardRing

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

Description

Opens the Create Guard Ring form, which lets you instantiate guard rings. If *w_windowId* is not specified, the current window is used.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCreateInst

```
leHiCreateInst(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Instance form, which lets you create an instance in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to specify the name of a library, a cell, and a view to use as the master cell for the instance. You can change the rotation of the instance or mirror the instance over the X or Y axis. You can also create an array of the instance. If the chosen cell is a Pcell, its parameters are added to the form.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateLabel

```
leHiCreateLabel(
     [ w_windowId ]
     [ t_mode ]
     )
     => t / nil
```

Description

Opens the Create Label form, which lets you create a label in the specified window. If *w_windowId* is not specified, the current window is used.

You can specify the text for the label, the height of the label, the font style, and the justification. You can choose options for *Drafting* and *Attach*. You can also rotate the label or mirror the label over the X or Y axis.

Arguments

w_windowId	ID of the window in which labels are to be created.
t_mode	Mode for creating the label. Valid values: `manual`, `fromConnectivity`

Values Returned

`t`	Returned when the Create Label form is already open.
`nil`	Returned when the command fails to run.

Examples

Opens the Create Label form for the current window:

```
leHiCreateLabel()
```

Opens the Create Label form for the current window:

```
leHiCreateLabel(hiGetCurrentWindow())
```

Opens the Create Label form for the current window with the *Manual* option selected:

```
leHiCreateLabel(hiGetCurrentWindow() "manual")
```

Prompts you to click the shape to be labeled. If you select a shape that does not have any associated net information, an appropriate warning message is generated in the CIW. You can press F3 to open the Create Label form to configure additional label settings.

```
leHiCreateLabel(hiGetCurrentWindow() "fromConnectivity")
```

leHiCreateMPP

```
leHiCreateMPP(
     [ w_windowId ]
     )
     => t / nil
```

Description

Runs the Create MPP command in the specified window. You can draw the MPP on the canvas or press `F3` to open the Create Multipart Path form and specify the settings for creating multipart paths. If `w_windowId` is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The multipart path was created.
`nil`	The multipart path was not created.

leHiCreatePath

```
leHiCreatePath(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Path form, which lets you create a path in the specified window. If *w_windowId* is not specified, the current window is used.

In manual mode you can set the width, offset, and justification of the path. You can set the snap mode, contact justification, end type, and begin and end extensions. You can also change the layer for the path.

If *Fixed Width* is on, the width remains the same. Otherwise, the `minWidth` property of the new layer determines the new width.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCreatePin

```
leHiCreatePin(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Pin form, which lets you create a geometric pin in the specified window. If *w_windowId* is not specified, the current window is used.

You can enter a name for the terminal, choose the *Mode*, choose the *I/O Type* and *Access Direction*, and set *Snap Mode*.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreatePinsFromLabels

```
leHiCreatePinsFromLabels(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Pins From Labels form, which lets you create pins from text labels in your layout cellview or selected instances. This command creates pins with terminal names matching labels on a specified text layer with pin dimensions that you specify, centered on the origin of your text label. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreatePlacementArea

```
leHiCreatePlacementArea(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the *Create – P&R Objects – Custom Placement Area* form, which lets you create a placement area in one of three modes: Manual, Assisted, or Auto. This form contains a *Dressing Template Editor* button to bring up the Dressing Template Editor subform. To activate this subform with SKILL, use the SKILL ~> operator like this: `leHiCreatePlacementArea~>Dressing~>value=1`. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCreatePolygon

```
leHiCreatePolygon(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Polygon form, which lets you create a polygon in the specified window. If *w_windowId* is not specified, the current window is used.

You are prompted to point to the coordinates for the vertices of the polygon. You can set the snap mode for the polygon and you can create polygons that contain arc segments.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreatePRBoundary

```
leHiCreatePRBoundary(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create P&R Boundary form, which lets you create a P&R boundary. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateRect

```
leHiCreateRect(
     [ w_windowId ]
     )
     => t / nil
```

Description

Creates a rectangle in the specified window. You are prompted to point to two coordinates for the opposite corners of the rectangle. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateRow

```
leHiCreateRow(
     [ w_windowId ]
     )
     => t / nil
```

Description

Makes sure the cellview is writable and opens the *Create – P&R Objects – Create Row* form, which lets you create a row in the specified window. Prerequisite: site definitions must already exist in your technology file. This form contains a *Dressing Template Editor* button to bring up the Dressing Template Editor subform. To activate this subform with SKILL, use the SKILL ~> operator like this: `leHiCreateRow~>Dressing~>value=1`.If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiCreateRuler

```
leHiCreateRuler(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Measurement form, which lets you create a ruler in the specified window. If *w_windowId* is not specified, the current window is used.

You can choose options to keep the ruler in the window after you enter the last point and create a multi-segment ruler that displays cumulative distance. You can also set the snap mode for the ruler.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateSnapBoundary

```
leHiCreateSnapBoundary(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Create Snap Boundary form, which lets you create a snap boundary. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateTaper

```
leHiCreateTaper(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Transmission Line Taper form, which lets you create a transmission line taper in the specified window. If *w_windowId* is not specified, the current window is used.

You can specify the *Taper Style*, *Resolution*, *Side Widths*, and *Snap Mode*.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateTrl

```
leHiCreateTrl(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Transmission Line form, which lets you create a transmission line in the specified window. If *w_windowId* is not specified, the current window is used.

You can set the bend style (bend, chamfer, or radial), the factors for the bend style, the resolution for the bend, and the width of the transmission line. You can also set the snap mode.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiCreateVia

```
leHiCreateVia(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Create – Via* command and opens the Create Via form, which lets you create a via in the specified window. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiDelete

```
leHiDelete(
     [ w_windowId ]
     )
     => t / nil
```

Description

Deletes selected objects in the specified window. If you delete a net selected using the *Navigator* assistant, all shapes on the net, except for pins, are also deleted. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiDeleteAllAreaViewLevel

```
leHiDeleteAllAreaViewLevel(
     [ w_windowId ]
     )
     => t / nil
```

Description

Removes all special display areas in window *w_windowId*. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiDeleteAreaViewLevel

```
leHiDeleteAreaViewLevel(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays all currently defined area view levels using the layer-purpose pair `hilight drawing1`, and prompts you to point at the area to be removed. If you do not specify *`w_windowId`*, the layout editor uses the current window.

Arguments

`w_windowId`	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiDeleteShapeFromNet

```
leHiDeleteShapeFromNet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Remove Shape From Net form, which lets you remove the selected shape from the net. If *w_windowId* is not specified, the current window is used.

You can delete selected shapes from the displayed net.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiDescend

```
leHiDescend(
     [ w_windowId ]
     [ t_mode ]
     )
     => t / nil
```

Description

Lets you push down one level into the design hierarchy. You can preselect the instance to descend into. If no instances are selected, you are prompted to select an instance. If you do not specify *w_windowId*, the layout editor uses the current window.

If the master cell of the selected instance has more than one cellview available, the function must determine which cellview to descend into. If the *Prompt For View Name* field is not set, or if there is only one view available, you descend into the cellview that has been instantiated. If more than one view is available, and the *Prompt For View Name* field is set, a form is displayed containing a cyclic field with all the possible view names listed. You can then select the cellview to descend into.

When the *t_mode* argument is not specified, automatically selects the edit mode to use when descending into the hierarchy. If the current cellview is opened in edit mode, the new cellview is opened in edit mode; if the new cellview cannot be opened in edit mode, it is opened in read mode. If the current cellview is opened in read mode, the new cellview is opened in read mode.

Arguments

w_windowId	ID for the specified window; defaults to current window.
t_mode	Optional text string specifying the mode in which to open the selected instance. When "`nil`", the instance is opened in the same mode as its parent. When "`read`", the instance is opened in read mode. When "`edit`", the instance is opened in edit mode if its parent is open in edit mode; otherwise, a window pops up stating that the instance cannot be opened in edit mode and advising that you open it in read mode. Valid Values: `edit`, `read`, `nil`.

Value Returned

`t`	The instance was opened successfully.

`nil`	The instance was not opened successfully.

leHiDisplayPadOpeningInfoForm

```
leHiDisplayPadOpeningInfoForm(
     )
```

Description

Opens the Pad Opening Info form, which you can use to find pad shapes at lower levels of your hierarchical cellview, generate reports and labels for the pad shapes that are found, and promote the generated labels to the top level of the cellview.

You can also access this form using the *Tools – Pad Opening Info* command.

Arguments

None

Value Returned

None

leHiDisplayTechGraphForm

```
leHiDisplayTechGraphForm(
     )
     => t
```

Description

Opens the Technology Database Graph form. You can also access this form using the *CIW – Tools – Technology File Manager – Graph* command.

Arguments

None

Value Returned

`t`	The function is successful.

leHiEditDisplayOptions

```
leHiEditDisplayOptions(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Display Options form, which lets you change the display options in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You can control the display of objects and the grid, how the cursor snaps to the grid, and the default snap mode settings for the create and edit forms.

These options correspond to window environment variables that can be initialized in the `.cdsenv` file or saved to the current cellview.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful

leHiEditDRDOptions

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

Description

Opens the DRD Options form, which lets you select the rules that are to be considered by the Design Rule Driven Editing. You can also set the other DRD options, such as mode, hierarchy depth, display, through the DRD Options form. Corresponds to the F7 bindkey in the `leBindkeys.il` file.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiEditDRDRuleOptions

```
leHiEditDRDRuleOptions(
     )
```

Description

Opens the DRD Options form, which lets you select the rules that are to be considered by the Design Rule Driven Editing. You can also set the other DRD options, such as mode, hierarchy depth, display, through the DRD Options form. Corresponds to the F7 bindkey in the `leBindkeys.il` file.

Arguments

None

Value Returned

None

leHiEditDynamicMeasurementOptions

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

Description

Opens the Dynamic Display form, which lets you change the dynamic measurement display options. This form turns dynamic measurement on or off, and sets which measurements will display, and the height of the text display in pixels.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiEditEditorOptions

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

Description

Opens the Layout Editor Options form, which lets you change the layout editor environment. You can also set these variables in your `.cdsenv` file, so the layout editor defaults to these options on start up.

You can control the behavior of various editing functions, the number of sides for conics, and the type of object the cursor snaps to. You can also set how close the cursor must be to an object before the cursor snaps (aperture), how many levels down in the hierarchy the cursor snaps (depth), and an offset distance for the cursor snap (bounce).

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiEditInPlace

```
leHiEditInPlace(
     [ w_windowId ]
     )
     => t / nil
```

Description

Lets you edit an instance's master while viewing it in the context of the instance placement. If you do not specify *w_windowId*, the layout editor uses the current window. You can preselect the instance to be edited in place. If no instance is selected, you are prompted to point at a shape contained in the instance to be edited in place. If you point at several shapes at different levels of the hierarchy, the shape in the higher-level instance is the one that is used.

You can change the window to any level in the hierarchy. For example, if you are already using `leHiEditInPlace` to edit a cellview two levels deep in your hierarchy, you can use `leHiEditInPlace` again to move up or down the hierarchy.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiEditProp

```
leHiEditProp(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Edit Properties form, which lets you edit the properties set in the specified window. You are prompted to select objects whose properties you want to edit. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiFlatten

```
leHiFlatten(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Flatten form, which lets you explode a cell instance or array, moving the contents of the cell or array up one or more levels into the current level of the hierarchy. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiFlip

```
leHiFlip(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the *Flip* form that you can use to flip an object horizontally or vertically in the specified window.

Arguments

w_windowId	Database ID of the window in which you want to start the command. If *w_windowId* is not specified, the current window is used.

Value Returned

t	The function is successful.
nil	The function is not successful.

Examples

Both the APIs open the *Flip* form in the current window.

```
leHiFlip(hiGetCurrentWindow())
leHiFlip()
```

leHiIncrementalViolation

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

Description

Specifies the incremental command mode.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

Examples

```
leHiIncrementalViolation()
```

leHiIncrementalViolationUpdater

```
leHiIncrementalViolationUpdater(
     )
```

Description

Enables you to iterate between the violations of the command.

Arguments

`t`	Iterates forward.
`nil`	Iterates backward.

Value Returned

None

Examples

```
leHiIncrementalViolationUpdater(t)
leHiIncrementalViolationUpdater(nil)
```

leHiLayerGen

```
leHiLayerGen(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Layer Generation form, which lets you create new shapes by performing logical operations on selected objects in the specified layers. If you do not specify *w_windowId*, the layout editor uses the current window.

You choose the first layer of the operation, the type of operation (`AND`, `OR`, `AND NOT`, `XOR`, or `GROW BY`), the second layer of the operation, and the layer on which the new shapes are created.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiLayerTap

```
leHiLayerTap(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Tools – Tap* command and opens the Tap form when Options Displayed When Commands Start is set to on. Tapping sets the entry layer to the layer of an indicated shape in the current window. You can also use the tap command to seed create forms with information such as nets and attributes of objects. You are prompted to point at a shape with attributes to be tapped. You can point to shapes anywhere in the hierarchy. If the shape you want to select is overlapped by other shapes, you can cycle through the shapes until the shape you want is displayed, by setting the environment variable `layerTapCycle` to `t`, which takes precedence over Select from overlaps. If `layerTapCycle` is `nil` and `layerTapPick` is `t`, then the Select from overlaps form will appear if you click more than one overlapped shape. If you do not point to any shapes that are on valid layers, the function calls `hiGetAttention` and prompts you again. If you do not specify *`w_windowId`*, the layout editor uses the current window.

Arguments

`w_windowId`	Window ID of the window specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiMakeCell

```
leHiMakeCell(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Make Cell form, which lets you create a cell from the selected objects in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to enter a cell name for the new cell. You can choose to replace the selected objects with the newly created cell.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiMerge

```
leHiMerge(
     [ w_windowId ]
     )
     => t / nil
```

Description

Merges selected objects that touch or overlap each other into one object. You can merge shapes of different types that are on the same layer. You cannot merge shapes on different layers. Pins cannot be merged. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiModifyCorner

```
leHiModifyCorner(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Modify Corner form, which lets you modify corners of rectangles and polygons in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to choose the corners you want to modify. You can specify the type of corner to create, the radius or distance to use, and how many sides to use. The radius or distance used is half the length of the longest line segment adjacent to the corner or the value supplied, whichever is shorter.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiMousePopUp

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

Description

Opens the layout editor Middle Mouse Button Pop Up form. It is context sensitive. Returns `t` if the current window is a layout window, returns `nil` if the current window is not a layout window.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiMove

```
leHiMove(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Move form, which lets you move an object in the specified window to another location or layer in the same window or in a different window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to choose the object you want to move. You can change the layer or snap mode or rotate or mirror the object.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiMoveOrigin

```
leHiMoveOrigin(
     [ w_windowId ]
     )
     => t / nil
```

Description

Enter this function with only the window ID argument; the system prompts you to point to the new origin. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiOptionLayer

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

Description

Opens the LSW Option form. Returns `t` when you click *OK*, returns `nil` when you click *Cancel*.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiPaste

```
leHiPaste(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Paste form, which lets you place the contents of the yank buffer in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to specify where you want the objects placed. You can choose to rotate or mirror the object.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiPlotQueueStatus

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

Description

Opens the Plot Job Queue form, which shows the status of the plot job queue.

You can view the printer queue and remove jobs from the plot job queue.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiPropagateNets

```
leHiPropagateNets(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Connectivity tab of the property editor for the selected instances. The tab lets you promote net information from pins in selected blocks one level down in the hierarchy up to the top level where the router can recognize and route the nets. You use this command when you do not have a schematic for your layout cellview. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiQuery

```
leHiQuery(
     [ w_windowId ]
     )
     => t / nil
```

Description

Invokes the Query command to initiate a query in which you can select a shape to have descriptive data about it displayed.

Arguments

w_windowId	The identifier of the window in which to execute the query. This option can be used to target a design window when multiple windows are open.

Value Returned

`t`	Returns `t` if the query is successfully initiated.
`nil`	Returns `nil` if the query is not successfully initiated.

leHiQuickAlign

```
leHiQuickAlign(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Quick Align* command in the specified window. You are prompted to select a reference edge, centerline, or point. In the post-select mode, you can select the reference edge, centerline, or point after selecting the reference set objects. You can press `F3` to open the *Quick Align* form and specify the settings for alignment.

Arguments

w_windowId	Database ID of the window in which you want to start the command. If *w_windowId* is not specified, the command starts in the current window.

Value Returned

`t`	Returned in the pre-select mode, after the command completes.
`nil`	Returned when you press `Esc` or click *Cancel* on the *Quick Align* form or when the command fails to start.

leHiReShape

```
leHiReShape(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Reshape form, which lets you modify the vertices of rectangles, polygons, paths, and transmission lines. You can add vertices, remove vertices, or modify vertex locations. The modified shape is stored in its most efficient form. For example, if a polygon is modified to have a rectangular shape, it is converted into a rectangle. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to choose the object you want to reshape. You can choose to use a rectangle or line as the reshape type, and you can choose the snap mode for creating the reshape type.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiRemasterInstances

```
leHiRemasterInstances(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Remaster Instances form, in which you specify instances to search for based on the view name, cell name, and the library name for a given cellviewID, then re-masters those instances with the master that matches the update library, update cell and update view names you provide in the form. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiRemoveFromGroup

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

Description

This function operates only when nested within the `leHiEditInPlace` function. After executing the leHiEditInPlace function, select the group from which you want to remove a figure, and then execute `leHiRemoveFromGroup`, which prompts you to select figures to remove from the current group. If you select all of the figures in the group, you can exit the function with an escape or by starting another function. However, if you select fewer figures than the group contains, you must exit Edit-In-Place mode with the leReturnToLevel function (*Edit – Hierarchy – Return To Level*).

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiRepeatCopy

```
leHiRepeatCopy(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Repeat Copy* command in the specified window.

Arguments

w_windowId	Database ID of the window in which you want to start the command. If *w_windowId* is not specified, the current window is used.

Value Returned

t	The command starts successfully.
nil	The command does not run.

Examples

Both the APIs start the *Repeat Copy* command in the current window.

```
leHiRepeatCopy(hiGetCurrentWindow())
leHiRepeatCopy()
```

leHiRotate

```
leHiRotate(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Rotate form, which lets you change the orientation of geometric objects. If *w_windowId* is not specified, the current window is used.

You choose the snap-to and rotation angle. You can also rotate objects 90, 180, or 270 degrees by using the buttons.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSave

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

Description

Executes the *File – Save* command, which saves the current cellview.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiSaveACopy

```
leHiSaveACopy(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Save As Copy form, which lets you copy the current cellview to another library, cellview name, and/or view name. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSaveHier

```
leHiSaveHier(
     [ w_windowId ]
     )
     => t / nil
```

Description

Enables you to save a hierarchical instance that is modified without running the *Edit In Place* or *Descend Edit* command.

Arguments

w_windowId	Database ID of the window in which you want to save the hierarchical instances that are modified without editing them in place. If *w_windowId* is not specified, the function starts in the current window.

Value Returned

t	Hierarchical instances are saved.
nil	Hierarchical instances are not saved.

Example

Saves the hierarchical instance that has been modified without editing it in place in the current window.

```
leHiSaveHier(currWindowId)
```

leHiSearch

```
leHiSearch(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Find/Replace form, which lets you search the specified window for objects that match user-specified query requests. If you do not specify *w_windowId*, the layout editor uses the current window.

You select the type of object to search for, how deep in the hierarchy to search, and the criteria types to search.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSetAreaViewLevel

```
leHiSetAreaViewLevel(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Set Area View Level form, which lets you create a special display area in the window. This lets you override the window's display start and stop levels for a specified area. If you do not specify *w_windowId*, the layout editor uses the current window.

The system prompts you to create the coordinates for the bounding box and type in the start and stop levels.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSetRefPoint

```
leHiSetRefPoint(
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets a reference point in the specified window. You are prompted to point to location for the reference point. If you do not specify *w_windowId*, the layout editor uses the current window.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiShowAngles

```
leHiShowAngles(
     [ w_windowId ]
     )
     => t / nil
```

Description

Runs the *Show Angles* command in the specified window. You are prompted to point to a vertex, an edge, or an object to view the angle formed at the intersection of two edges. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiShowCoords

```
leHiShowCoords(
     [ w_windowId ]
     )
     => t / nil
```

Description

Runs the *Show Coordinates* command in the specified window. You are prompted to point to a vertex, an edge, or an object to view the coordinates of the selected shape. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiShowSelSet

```
leHiShowSelSet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays information about the selected set in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

The Show Selected Set list displays information about all objects in the selected set: the type of shape, the layer and purpose, and the coordinates used to create the object. If instances or contacts are selected, the list displays the master name, instance name and type, orientation, and origin.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSize

```
leHiSize(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Size form, which lets you enlarge or reduce a shape in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to enter a value for the size. A positive number enlarges an object, and a negative number reduces an object.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSplit

```
leHiSplit(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Split form, which lets you reshape an object by stretching part of it relative to a split line in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to create the split line and then stretch the newly created edges.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiStretch

```
leHiStretch(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Stretch form, which lets you stretch objects in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to select an object to stretch. You can change the snap mode or lock angles when you stretch an object.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiSubmitPlot

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

Description

Opens the Submit Plot form, which lets you submit plots to a plotter.

You can submit plots to a plotter or file and specify the area to plot, when to plot, how big to make the plot, and how objects appear in the plot.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiSummary

```
leHiSummary(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays information about the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

The summary file displays the following information:

- The cellview environment
- Layer Object Statistics
- Instance Statistics
- Wire Statistics
- Marker Object Statistics
- Row Object Statistics
- Track Pattern Statistics
- Blockage Object Statistics
- Snap Boundary Object Statistics
- PR Boundary Object Statistics
- Area Boundary Object Statistics
- Cluster Boundary Object Statistics

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiTree

```
leHiTree(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays information about the design hierarchy in the specified window. If you do not specify *w_windowId*, the layout editor uses the current window.

You can choose the order of the display (top to bottom, current to bottom, or top to current) from the Tree form. The default is *Current to bottom*. The design hierarchy shows all placed instances in the window, listing the library, cell, and view names and the number of occurrences of each cell.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiUngroup

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

Description

Starts the *Edit – Group – Ungroup* command and prompts you to point at the group you want to ungroup.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

leHiYank

```
leHiYank(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Yank Form, which lets you copy a set of objects in a cellview into the yank buffer for later pasting. A yank shape defines the area enclosing the objects to copy. If you do not specify *w_windowId*, the layout editor uses the current window.

You are prompted to enter the coordinates of the yank shape.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leiDiscardEdits

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

Description

Discards all edits you made since the last time you saved. Corresponds to the cellview window *Design – Discard Edits* command.

A dialog box will appear asking you to confirm that you want to discard edits.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

lePinModelInitFunction

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

Description

Opens the Pin Connectivity Model graphical user interface, which you can use to set the connectivity model for the pins and terminals in your design.

Arguments

None

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

updateReturnPopupMenuItem

```
updateReturnPopupMenuItem(
     s_popUpMenu
     w_windowId
     )
     => t / nil
```

Description

Adds the Return menu item to the pop up menu specified.

Arguments

s_popUpMenu	Pop up menu identifier
w_windowId	Database ID of the specified window

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

Adds Return menu item to `lePopUpMenu`.

```
updateReturnPopoupMenuItem(lePopUpMenu window(1))
```

Start Router Functions

You can use SKILL functions to start the Virtuoso Chip Assembly Router.

- `leHiDisplayStartRouterForm`: Opens the Virtuoso Chip Assembly Router form.
- `leStartRouter`: Starts the Virtuoso Chip Assembly Router with specified data and startup options.

Related Topics

leHiDisplayStartRouterForm

leStartRouter

leHiDisplayStartRouterForm

```
leHiDisplayStartRouterForm(
     [ ?layoutLCV list(t_library t_cell t_view) ]
     )
     => t
```

Description

Opens the Virtuoso Chip Assembly Router form.

Arguments

`?layoutLCV list(`*`t_library t_cell t_view`*`)`

The library, cell, and view to route.

Value Returned

`t` The command executed successfully.

Example

Displays Start Router form for library `lib`, cell `top`, and view `layout`.

```
leHiDisplayStartRouterForm('("lib" "top" "layout"))
```

leStartRouter

```
leStartRouter(
     [ ?layoutLCV list(t_libraryName t_cellName t_viewName) ]
     [ ?layoutRoutedLCV list(t_libraryName t_cellName t_viewName) ]
     [ ?constraintGroup t_constraintGroupName ]
     [ ?doFile S_doFile ]
     [ ?initCommand S_initCommand ]
     [ ?showGraphics g_showGraphics ]
     [ ?quitAtFinish g_quitAtFinish ]
     [ ?useStartupFiles g_useStartupFiles ]
     [ ?usePreroutes g_usePreroutes ]
     [ ?usePreroutedIO g_usePreroutedIO ]
     [ ?stripOrphanShapes g_stripOrphanShapes ]
     [ ?checkAtStartup g_checkAtStartup ]
     [ ?cutKeepoutByClearance g_cutKeepoutByClearance ]
     [ ?cutKeepoutToImageEdge g_cutKeepoutToImageEdge ]
     [ ?createDidFile g_createDidFile ]
     [ ?didFile S_didFile ]
     [ ?messageFile S_messageFile ]
     [ ?statusFile S_statusFile ]
     [ ?startupOptions S_startupOptions ]
     )
     => t / nil
```

Description

Starts the Virtuoso Chip Assembly Router with specified data and startup options.

Arguments

`?layoutLCV list` *`(t_libraryName t_cellName t_viewName)`*

The name of the library, cell, and view to be saved to the router.

`?layoutRoutedLCV list` *`(t_libraryName t_cellName t_viewName)`*

The name of the library, cell, and view to be saved to the router. Allows you to choose a different name for the library, cell, and view to be saved to the router.

`?constraintGroup` *`t_constraintGroupName`*

. Name of the constraint group.

`?doFile` *`S_doFile`* Specifies a file with routing instructions that runs at the beginning of the session.

`?initCommand` *`S_initCommand`*

Lets you enter commands that you want to perform at the beginning of the session.

`?showGraphics` *`g_showGraphics`*

Controls whether the router runs with or without the Graphical User Interface (GUI). Turn off this option to prevent graphics from being displayed. Default is `t`.

`?quitAtFinish` *`g_quitAtFinish`*

Exits the session after running the do file. Default is `nil`.

`?useStartupFiles` *`g_useStartupFiles`*

Controls whether the router reads color map information and key definitions saved in the colors and keys files in your `.cct` directory. Default is `t`.

`?usePreroutes` *`g_usePreroutes`*

Controls whether the router displays and allows editing of prerouted wires defined in the design file. Turn off this option if you want to ignore prerouted wires. Default is `t`.

`?usePreroutedIO` *`g_usePreroutedIO`*

Controls whether the router displays and allows routing to prerouted I/O ports defined in the design file. Turn off this option if you want to ignore prerouted I/O ports. Default is `nil`.

`?stripOrphanShapes` *`g_stripOrphanShapes`*

Controls whether the router removes (the default) or retains orphan shapes that are defined in the design but not assigned to a net. Default is `t`.

`?checkAtStartup` *`g_checkAtStartup`*

Displays a list of invocation errors during startup. Default is `t`.

`?cutKeepoutByClearance` *`g_cutKeepoutByClearance`*

Controls whether the router automatically cuts clearance areas for pins enclosed in image keepouts. Default is `nil`.

`?cutKeepoutToImageEdge` *`g_cutKeepoutToImageEdge`*

Controls whether the router automatically cuts paths for pins enclosed in image keepouts. Default is `nil`.

`?createDidFile` *`g_createDidFile`*

Controls whether the router logs commands you use during the session in a did file. Default is `t`.

`?didFile` *`S_didFile`*

The name of the did to be used.

`?messageFile` *`S_messageFile`*

Sends prompts and messages to the specified file, in addition to sending them to the output window.

`?statusFile` *`S_statusFile`*

Name of the file where routing status information is saved.

`?startupOptions` *`S_startupOptions`*

Allows you to enter any or all startup options, such as `-lefdef`, `-nets`, `-do`, and `-abstracts` when launching the router.

Value Returned

`t`	The operation was successful.
`nil`	The operation was not successful.

Example

```
leStartRouter(
?layoutLCV list("lib" "top" "layout")
?showGraphics nil
?quitAtFinish t
?initCommand "check (type all)"
?useStartupFiles nil
?usePreroutes nil
?usePreroutedIO t
?stripOrphanShapes nil
?checkAtStartup nil
?cutKeepoutByClearance t
?cutKeepoutToImageEdge t
?createDidFile t
?didFile "did.did"
?messageFile "message.out"
?statusFile "status.sts"
?startupOptions "-lefdef"  )
```

The following example starts the Virtuoso Chip Assembly Router on `lib/cell/layout` to create `lib/cell/layout.routed` using the constraint group `virtuosoDefaultSetup`.

```
leStartRouter(
?layoutLCV list("lib" "cell" "layout")
?layoutRoutedLCV list("lib" "cell" "layout.routed")
?createDidFile nil
?constraintGroup "virtuosoDefaultSetup"  )
```

Via Functions

Use the Via SKILL functions to create and manage vias. For example, you can use `viaGenerateViasAtPoint` to generate vias with specified options at a given location in a window or cellview. You can use `viaLoadViaVariants` to load via variants from a specified cellview for the Create Via command.

Related Topics

Via Creation

viaFindTransitions

viaGenerateViasAtPoint

viaGenerateViasFromShapes

viaGenerateViasInArea

viaFindTransitions

```
viaFindTransitions(
     t_constraintGroupSetup
     t_topLayerName
     t_bottomLayerName
     )
     => l_viaTransitionsObject / nil
```

Description

Returns the list of transitions of the given constraint group setup object found between the specified top and bottom layers.

Arguments

t_constraintGroupSetup

This object is retrieved as follows:

```
viaOptions = viaGetOptions(constraintGroupId)
constraintGroupSetup = viaOptions~>constraintGroupSetup
```

t_topLayerName The name of the top layer.

t_bottomLayerName The name of the bottom layer.

Value Returned

l_viaTransitionsObject

Returns a list of via transitions objects representing the transition between the specified top layer and bottom layer.

`nil` Returns `nil` if the top layer name or bottom layer name or both are not found in the `t_constraintGroupSetup` object or if there is no transition path between the top and bottom layers.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
```

```
transitionList = viaFindTransitions(myViaOptions~>constraintGroupSetup "Metal4"
"Metal1")


transitionList~>??
    ((topLayer "Metal4" bottomLayer "Metal3" transitionEnabled
    t viaDef "M4_M3" cutClass "Auto"
    cutClassOrientation "auto" userCutSpacingEnabled nil userCutSpacingX
    0.0 userCutSpacingY 0.0
    )
     (topLayer "Metal3" bottomLayer "Metal2" transitionEnabled
    t viaDef "M3_M2" cutClass "Auto"
    cutClassOrientation "auto" userCutSpacingEnabled nil userCutSpacingX
    0.0 userCutSpacingY 0.0
     )
     (topLayer "Metal2" bottomLayer "Metal1" transitionEnabled
    t viaDef "M2_M1" cutClass "Auto"
    cutClassOrientation "auto" userCutSpacingEnabled nil userCutSpacingX
    0.0 userCutSpacingY 0.0
     )
     )
```

viaGenerateViasAtPoint

```
viaGenerateViasAtPoint(
     w_windowId | d_cellViewId
     l_point
     t_viaOptionsObject
     [ ?topAndBottomLayers lt_topAndBottomLayers ]
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     )
     => l_via / nil
```

Description

Generates vias with the specified options at a given location in the specified window or cellview.The figures present at the given location are queried, and if they form valid overlaps, vias are created to connect the figures.

Arguments

w_windowId | *d_cellViewId*

Window ID or cellview ID. If window ID is specified, the vias are generated in the edited cellview opened in the given window ID. In addition, if a window ID is given, the window settings are respected (layer visibility, edited figGroup, window start and stop level, and selectCreatedObj environment variable).

l_point

List that represents the point where the vias are to be generated.

t_viaOptionsObject

The options used to customize the via generation.

?topAndBottomLayers *lt_topAndBottomLayers*

Forces the creation of the via between the specified top and bottom layer names even if the via overlaps top and bottom layers that are computed are different. If this argument is not specified, the top and bottom layers of the overlaps computed at the given location are used.

?startLevel *x_startLevel*

Specifies the start level of the query run to compute the via overlaps. If the start level is not specified, and a window ID is specified as the first argument of the function, the window~>startLevel is used. If the start level is not specified and a cellview ID is specified as the first argument of the function, `0` is used as the value of the start level.

`?stopLevel` *`x_stopLevel`*

Specifies the stop level of the query run to compute the via overlaps. If the stop level is not specified, and a windowID is specified as the first argument of the function, the window~>stopLevel is used. If the stop level is not specified and a cellview ID is specified as the first argument, `0` is used as the value of the stop level.

Value Returned

`l_via`	Returns the list of the created vias.
`nil`	Returns `nil` if no via can be created at the specified location.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
window = hiGetCurrentWindow()

viaGenerateViasAtPoint(window list(0 0) myViaOptions)
    (db:0x2550d41a)
```

viaGenerateViasFromShapes

```
viaGenerateViasFromShapes(
     w_windowId | d_cellViewId
     l_shapes
     t_viaOptionsObject
     [ ?topAndBottomLayers lt_topAndBottomLayers ]
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     )
     => l_via / nil
```

Description

Generates vias from the specified shapes with the given via options in the specified window or cellview. If the specified shapes form valid overlaps, vias are created to connect the shapes.

Arguments

w_windowId | *d_cellViewId*

Window ID or cellview ID. If window ID is specified, the vias are generated in the edited cellview opened in the given window ID. In addition, if a window ID is given, the window settings are respected (layer visibility, edited figGroup, window start and stop level, and selectCreatedObj environment variable).

l_shapes

List of shapes used to compute the overlaps needed to generate the vias.

t_viaOptionsObject

The options used to customize the via generation.

?topAndBottomLayers *lt_topAndBottomLayers*

Forces the creation of the via between the specified top and bottom layer names even if the via overlaps top and bottom layers that are computed are different. If this argument is not specified, the top and bottom layers of the overlaps are used.

?startLevel *x_startLevel*

Specifies the start level of the query run to compute the via overlaps. If the start level is not specified, and a window ID is specified as the first argument of the function, the window~>startLevel is used. If the start level is not specified and a cellview ID is specified as the first argument of the function, `0` is used as the value of the start level.

`?stopLevel` *`x_stopLevel`*

Specifies the stop level of the query run to compute the via overlaps. If the stop level is not specified, and a windowID is specified as the first argument of the function, the window~>stopLevel is used. If the stop level is not specified and a cellview ID is specified as the first argument, `0` is used as the value of the stop level.

Value Returned

`l_via`	Returns the list of created vias.
`nil`	Returns `nil` if no via can be created to connect the specified shapes.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
cv = deGetCellView()

viaGenerateViasFromShapes(cv cv~>shapes myViaOptions)
    (db:0x172ed51a db:0x172ed51b db:0x172ed51c db:0x172ed51d db:0x172ed51e)
```

viaGenerateViasInArea

```
viaGenerateViasInArea(
     w_windowId | d_cellViewId
     l_ptArray
     t_viaOptionsObject
     [ ?topAndBottomLayers lt_topAndBottomLayers ]
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     )
     => l_via / nil
```

Description

Generates vias with the specified options within a given area in the specified window or cellview. The figures present at the given location are queried, and if they form valid overlaps, vias are created to connect the figures.

Arguments

w_windowId | *d_cellViewId*

Window ID or cellview ID. If window ID is specified, the vias are generated in the edited cellview opened in the given window ID. In addition, if a window ID is given, the window settings are respected (layer visibility, edited figGroup, window start and stop level, and selectCreatedObj environment variable).

l_ptArray

List that represents the area where the vias are to be generated. The area specified can be polygonal or rectangular. For a rectangular area, four points should be specified for the array.

t_viaOptionsObject

The options used to customize the via generation.

?topAndBottomLayers *lt_topAndBottomLayers*

Forces the creation of the via between the specified top and bottom layer names even if the via overlaps top and bottom layers that are computed are different. If this argument is not specified, the top and bottom layers of the overlaps computed at the given location are used.

?startLevel *x_startLevel*

Specifies the start level of the query run to compute the via overlaps. If the start level is not specified, and a window ID is specified as the first argument of the function, the window~>startLevel is used. If the start level is not specified and a cellview ID is specified as the first argument of the function, `0` is used as the value of the start level.

`?stopLevel` *`x_stopLevel`*

Specifies the stop level of the query run to compute the via overlaps. If the stop level is not specified, and a windowID is specified as the first argument of the function, the window ~>stopLevel is used. If the stop level is not specified and a cellview ID is specified as the first argument, `0` is used as the value of the stop level.

Value Returned

`l_via`	Returns the list of created vias.
`nil`	Returns `nil` if no via can be created at the specified location.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
myViaOptions~>createInRoute = nil
cv = deGetCellView()

viaGenerateViasInArea(cv list('(-20 -20) '(-20 20) '(20 20) '(20 -20))
myViaOptions)
    (db:0x2dc9d41a db:0x2dc9d41b)
```

viaGetViaOptions

```
viaGetViaOptions(
     d_ConstraintGroupID
     )
     => t_viaOptionsObject / nil
```

Description

Returns a via options object from the specified constraint group. This via options object can be used by any SKILL procedure enabling generation or re-computation of vias.

The via options object gathers all the options allowing to customize the viaEngine during the via generation or re-computation. All these options are pre-seeded with the default environment variable values. On the other hand, in the createVia form, the options are pre-seeded with the current environment variable values.

The available settings for UserCutSpacing in the via options are:

viaOptions~>constraintGroupSetup~>transition~>userCutSpacingEnabled

viaOptions~>constraintGroupSetup~>transition~> userCutSpacingX

viaOptions~>constraintGroupSetup~>transition~> userCutSpacingY

Arguments

d_ConstraintGroupID

The ID of the constraint group in which the constraint is searched.

Value Returned

t_viaOptionsObject

Returns the via options object name. The following are some via options for via fill:

maxNumOverlaps: Specifies the maximum number of overlaps on which to create vias.

```
myViaOptions->automatic~>maxNumOverlaps = 100
```

viaMaskColor: When a single transition is enabled, this option specifies the mask on which to color the cut layer of the transition. The available values are between 1 for “mask1Color” and 8 for “mask8Color”.

```
myViaOptions->automatic~>viaMaskColor = 2
```

validMaskColorsPerOverlapLayer: Specifies the allowed mask colors for a given layer. Shapes on the given layer but another mask color are not considered for determination of the overlaps.

```
myViaOptions->automatic~>validMaskColorsPerOverlapLayer
= "Metal1 1 2"
```

The example restricts shapes on `Metal1` to be on `mask1Color` or `mask2Color` for being considered in overlaps calculation.

`nil` Returns `nil` if the constraint group is not found or is not a valid `cst` constraint group.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile
"virtuosoDefaultSetup")

myViaOptions = viaGetViaOptions(constraintGroupId)
```

```
myViaOptions~>??
    (createInRoute t createAsROD nil rodName
         "" automatic via:0x3be390d0 constraintGroupSetup via:0x3be39030
     )
```

viaLoadViaVariants

```
viaLoadViaVariants (
     d_cellViewId
     )
     => t
```

Description

Loads the via variants from the specified cellview for the Create Via command.

Arguments

d_cellViewId	The cellview ID from which the via variants are loaded.

Value Returned

t	Returns t if via variants are loaded.

Example

```
viaLoadViaVariants(cvId)
```

viaRecomputeVias

```
viaRecomputeVias(
     w_windowId | d_cellViewId
     l_via
     t_viaOptionsObject
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     )
     => l_via / nil
```

Description

Recomputes the via list with the given via options in the specified window or cellview. The figures connected by the vias are searched and the vias are recomputed with the via options specified to connect the figures.

Arguments

w_windowId | *d_cellViewId*

Window ID or cellview ID. If window ID is specified, the vias are generated in the edited cellview opened in the given window ID. In addition, if a window ID is given, the window settings are respected (layer visibility, edited figGroup, window start and stop level, and selectCreatedObj environment variable).

l_via

List of vias to recompute.

t_viaOptionsObject

The options used to customize the via recomputation.

?startLevel *x_startLevel*

Specifies the start level of the query run to retrieve the figure connected by the vias. If the start level is not specified, and a window ID is specified as the first argument of the function, the window ~>startLevel is used. If not specified and a cellview ID is specified as the first argument, `0` is used as the value of the start level.

?stopLevel *x_stopLevel*

Specifies the stop level of the query run to retrieve the figures connected by the vias. If the stop level is not specified, and a windowID is specified as the first argument of the function, the window~>stopLevel is used. If the stop level is not specified and a cellview ID is specified as the first argument, `0` is used as the value of the stop level.

Value Returned

`l_via`	Returns the list of modified and created vias.
`nil`	Returns `nil` if no via can be recomputed.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
cv = deGetCellView()
myViaOptions~>automatic~>extendEnclosureBeyondOverlap = nil
myViaOptions~>automatic~>extendEnclosureToOverlap = nil
viaRecomputeVias(cv cv~>vias myViaOptions)
    (db:0x17b0d19c db:0x17b0d19d)
```

viaRecomputeViasAtPoint

```
viaRecomputeViasAtPoint(
     w_windowId | d_cellViewId
     l_point
     t_viaOptionsObject
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     )
     => l_via / nil
```

Description

Recomputes vias with the specified options at a given location in the specified window or cellview. The vias present at the given location are queried along with the figures connected by the vias. The vias are then recomputed with the via options specified to connect the figures.

Arguments

w_windowId | *d_cellViewId*

Window ID or cellview ID. If window ID is specified, the vias are generated in the edited cellview opened in the given window ID. In addition, if a window ID is given, the window settings are respected (layer visibility, edited figGroup, window start and stop level, and selectCreatedObj environment variable).

l_point

List that represents the point where the via is to be recomputed.

t_viaOptionsObject

The options used to customize the via recomputation.

?startLevel *x_startLevel*

Specifies the start level of the query run to retrieve the vias and the figures connected by the vias. If the start level is not specified, and a window ID is specified as the first argument of the function, the window~>startLevel is used. If the start level is not specified and a cellview ID is specified as the first argument of the function, `0` is used as the value of the start level.

?stopLevel *x_stopLevel*

Specifies the stop level of the query run to retrieve the vias and the figures connected by the vias. If the stop level is not specified, and a windowID is specified as the first argument of the function, the window~>stopLevel is used. If the stop level is not specified and a cellview ID is specified as the first argument, `0` is used as the value of the stop level.

Value Returned

`l_via`	Returns the list of modified and created vias.
`nil`	Returns `nil` if no via is found at the given location.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
cv = deGetCellView()

viaRecomputeViasAtPoint(cv list(253 52) myViaOptions)
    (db:0x271ed19a)
```

viaRecomputeViasInArea

```
viaRecomputeViasInArea(
     w_windowId | d_cellViewId
     l_ptArray
     t_viaOptionsObject
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     )
     => l_via / nil
```

Description

Recomputes vias with the specified options within a given area in the specified window or cellview. The vias present at the given location are queried along with the figures connected by the vias. The vias are then recomputed with the via options specified to connect the figures.

Arguments

w_windowId | *d_cellViewId*

Window ID or cellview ID. If window ID is specified, the vias are generated in the edited cellview opened in the given window ID. In addition, if a window ID is given, the window settings are respected (layer visibility, edited figGroup, window start and stop level, and selectCreatedObj environment variable).

l_ptArray

List that represents the area where the via is to be recomputed.

t_viaOptionsObject

The options used to customize the via recomputation.

?startLevel *x_startLevel*

Specifies the start level of the query run to retrieve the vias and the figures connected by the vias. If the start level is not specified, and a window ID is specified as the first argument of the function, the window~>startLevel is used. If the start level is not specified and a cellview ID is specified as the first argument of the function, `0` is used as the value of the start level.

?stopLevel *x_stopLevel*

Specifies the stop level of the query run to retrieve the vias and the figures connected by the vias. If the stop level is not specified, and a windowID is specified as the the first argument of the function, the window~>stopLevel is used. If the stop level is not specified and a cellview ID is specified as the first argument, `0` is used as the value of the stop level.

Value Returned

l_via	Returns the list of modified and created vias.
`nil`	Returns `nil` if no via is found at the given location.

Example

```
techFile = techGetTechFile(ddGetObj("gpdk090"))
constraintGroupId = cstFindConstraintGroupIn(techFile "virtuosoDefaultSetup")
myViaOptions = viaGetViaOptions(constraintGroupId)
cv = deGetCellView()
transitionM1M2 = car(viaFindTransitions(myViaOptions~>constraintGroupSetup
"Metal2" "Metal1"))
transitionM1M2~>userCutSpacingEnabled = t
transitionM1M2~>userCutSpacingX = 0.3
transitionM1M2~>userCutSpacingY = 0.3

viaRecomputeViasInArea(cv list('(-20 -10) '(-20 10) '(10 10) '(10 -25) '(-7.5 -25)
'(-7.5 -10)) myViaOptions)
    (db:0x2d57d19c db:0x2d57d19d)
```

viaRegisterCreateViaInitCallback

```
viaRegisterCreateViaInitCallback(
     t_functionName
     )
     => t / nil
```

Description

Registers the function to be called each time the Create Via form is opened. The registered function name should be a valid SKILL procedure and the init procedure that is registered should be defined without arguments.

Arguments

t_functionName	Name of the function to register for use in the Create Via form.

Value Returned

`t`	Returns `t` if the function is successfully registered.
`nil`	Returns `nil` if the function is not successfully registered.

Example

```
viaRegisterCreateViaInitCallback("myInitProc")
```

viaRegisterPostViaEngineCallback

```
viaRegisterPostViaEngineCallback(
     t_functionName
     )
     => t / nil
```

Description

Registers the function name to be called after the viaEngine processing. The registered function name should be a valid SKILL procedure. The postViaEngine procedure that is registered should be declared with one argument, the `viaEngineContext`. This argument is a user type representing the context of the viaEngine call. The `viaRegisterPostViaEngineCallback` function that is registered should be declared with the `viaEngineContext` argument. This argument is a user type representing the context of the viaEngine call. The function returns `t` or `nil`. If `nil` is returned by the `viaRegisterPostViaEngineCallback` function registered, vias are not created.

Arguments

t_functionName	Name of the function to register after the viaEngine processing.

Value Returned

`t`	Returns `t` if the function is successfully registered.
`nil`	Returns `nil` if the function is not successfully registered.

Example

```
procedure(myPostViaEngineProc(veContext)
println(veContext~>??)
t
)
viaRegisterPostViaEngineCallback("myPostViaEngineProc")
```

viaRegisterPostViaServerCallback

```
viaRegisterPostViaServerCallback(
     t_functionName
     )
     => t / nil
```

Description

Registers the function name to be called after the via server processing. The function should be a valid SKILL procedure. The `postViaServer` procedure that is registered should be declared with three arguments, the `constraintGroup`, the `overlapInfo`, and the list of `viaViaDef` user type computed by the via server. The `postViaServer` procedure returns a list of `viaViaDef` user types. You can update all `viaViaDefs` specified in the `viaViaDef` argument. You also have the option to modify the sequence of the `viaViaDef` list.

Arguments

t_functionName	The name of the function to register after the via server processing.

Value Returned

`t`	The function is successfully registered.
`nil`	The function is not successfully registered.

Example

```
procedure(myPostViaServerProc(constraintGroup overlapInfo viaParams)
   println(constraintGroup ~>??)
   println(overlapInfo)
   println(viaParams)

   viaParams ; returned value
)
viaRegisterPostViaServerCallback("myPostViaServerProc")
```

viaRegisterPreViaEngineCallback

```
viaRegisterPreViaEngineCallback(
     t_functionName
     )
     => t / nil
```

Description

Registers the function name to be called before via engine processing. The registered function name should be a valid SKILL procedure. The preViaEngine procedure that is registered should be declared with one argument, `viaEngineContext`. This argument is a user type representing the context of the viaEngine call. The `viaRegisterPreViaEngineCallback` function that is registered should be declared with the `viaEngineContext` argument. This argument is a user type representing the context of the viaEngine call. The function returns `t` or `nil`. If `nil` is returned by the `viaRegisterPreViaEngineCallback` function registered, vias are not created.

Arguments

t_functionName	Name of the function to register before the viaEngine processing.

Value Returned

`t`	Returns `t` if the function is successfully registered.
`nil`	Returns `nil` if the function is not successfully registered.

Example

```
procedure(myPreViaEngineProc(veContext)
println(veContext~>??)
t
)
viaRegisterPreViaEngineCallback("myPreViaEngineProc")
```

viaSaveViaVariants

```
viaSaveViaVariants(
     d_cellViewId
     )
     => t
```

Description

Saves the via variants from the Create Via command into the specified cellview.

Arguments

d_cellViewId	The cellview ID in which the via variants are saved.

Value Returned

t	Returns t if via variants are saved.

Example

```
viaSaveViaVariants(cvId)
```

viaSetDefaultCutClasses

```
viaSetDefaultCutClasses(
     t_techLibName
     [ t_constraintGroupName ]
     ((bottomLayer topLayer cutClass) ...)
     [ g_reset ]
     )
     => t / nil
```

Description

Sets the default cut classes to be used for each layer.

Arguments

t_techLibName	Name of the technology library.
t_constraintGroupName	Name of the constraint group. This is an optional argument.
((bottomLayer topLayer cutClass) ...)	Specify the names of the bottom layer, top layer, and the cut class.
g_reset	Optional boolean argument specifying whether previous data is to be reset or appended. The default is `t`, which means previous data is reset. When set to `nil`, previous data is appended.

Value Returned

`t`	Returns `t` if the default cut classes have been set.
`nil`	Returns `nil` if the cut classes have been set.

Examples

```
viaSetDefaultCutClasses(
"gpdk032"
"foundry"
list('("Poly" "Metal1" "Cont_square") '("Oxide" "Metal1" "Cont_rect") '("Metal2"
"Metal3" "Via2_large" ) '("Metal4" "Metal5" "Via4_square"))
)
```

Or

```
viaSetDefaultCutClasses(
"gpdk032"
"foundry"
list('("Poly" "Metal1" "Cont_square" ) '("Oxide" "Metal1" "Cont_rect")))


viaSetDefaultCutClasses(
"gpdk032"
"foundry"
list('("Metal2" "Metal3" "Via2_large" ) '("Metal4" "Metal5" "Via4_square"))
nil)
```

viaSetDefaultValidViaDefs

```
viaSetDefaultValidViaDefs(
     t_techLibName
     t_constraintGroupName
     l_viaDef
     [ g_reset ]
     )
     => t / nil
```

Description

Sets the default vias to be used by Wire Edit and Create Via commands from the list of valid vias of the specified constraint group.

Arguments

t_techLibName	Name of the technology library.
t_constraintGroupName	Name of the constraint group of the technology library.
l_viaDef	List of via definition names. Each via definition should exist in the `validVias` constraint of the constraint group.
g_reset	Optional boolean argument specifying whether to replace any existing list for the given constraint group, or keep the existing settings. If two valid via definitions are specified for the same top and bottom layers, the alphabetical order is honored.

Value Returned

`t`	Returns `t` if the default via definitions have been set. If a via definition is invalid, a message is printed on the CIW.
`nil`	Returns `nil` if the technology library or the constraint group could not be found.

Examples

Constraint Group:

```
;( group [override] )
;( ---- --------- )
```

```
( "virtuosoDefaultSetup" nil
interconnect(
( validLayers   (Metal1  Metal2  Metal3) )
( validVias     (M2_M1_A  M2_M1_B M3_M2_A M3_M2_B) )
) ;interconnect
) ;virtuosoDefaultSetup
```

The following function defines vias `M2_M1_B` and `M3_M2_B` to be used by default by the Wire Editor during the via stitching operation and the Create Via command:

```
viaSetDefaultValidViaDefs("testLib" "virtuosoDefaultSetup" list("M2_M1_B"
"M3_M2_B"))
```

viaSetValidTransitions

```
viaSetValidTransitions(
     [ ?techLibName t_techLibName ]
     [ ?constraintGroupName t_constraintGroupName ]
     (( topLayer bottomLayer g_enable) ...)
     [ g_reset ]
     )
     => t / nil
```

Description

Enables or disables the list of layer transitions. By default, all the transitions are enabled.

Arguments

`?techLibName` *t_techLibName*

Specifies the name of the technology library.

`?constraintGroupName` *t_constraintGroupName*

Specifies the name of the constraint group.

((topLayer bottomLayer g_enable) ...)

Specifies the names of the top layer, bottom layer, and the state, enabled or disabled, of the transition between the layers.

g_reset — Specifies if the previous data is to be reset or appended. The default is `nil`, which means previous data is appended. When set to `t`, previous data is reset.

Value Returned

`t` — The list of layer transitions has been enabled.

`nil` — Unable to enable the list of layer transitions due to an error.

Examples

Enables the list of layer transitions between the specified layers.

```
ls=list(list("Metal1" "Oxide" nil) list("Metal1" "Poly" t) list("Metal2" "Metal1"
t) list("Metal3" "Metal2" t))
viaSetValidTransitions("gpdk045" "foundry" ls)
```

or

Enables the list of layer transitions between the specified layers and resets the previous data.

```
reset=t
viaSetValidTransitions("gpdk045" "foundry" list('("Metal2" "Metal1" nil)
'("Metal4" "Metal3" t)) reset)
```

viaUnregisterCreateViaInitCallback

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

Description

Unregisters the createVia initialization procedure if it has been registered. If it has not been registered, this function does nothing.

Arguments

None

Value Returned

`t`	Returns `t` if the function is successfully unregistered.
`nil`	Returns `nil` if the function is not successfully unregistered.

Example

```
viaUnregisterCreateViaInitCallback()
```

viaUnregisterPostViaEngineCallback

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

Description

Unregisters the postViaEngine procedure if it has been registered. If it has not been registered, this function does nothing.

Arguments

None

Value Returned

`t`	Returns `t` if the function is successfully unregistered.
`nil`	Returns `nil` if the function is not successfully unregistered.

Example

```
viaUnregisterPostViaEngineCallback()
```

viaUnregisterPostViaServerCallback

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

Description

Unregisters the `postViaServer` procedure if it has been registered.

Arguments

None

Value Returned

`t`	The function is successfully unregistered.
`nil`	The function is not successfully unregistered.

Example

```
viaUnregisterPostViaServerCallback()
```

viaUnregisterPreViaEngineCallback

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

Description

Unregisters the preViaEngine procedure if it has been registered. If it has not been registered, this function does nothing.

Arguments

None

Value Returned

`t`	Returns `t` if the function is successfully unregistered.
`nil`	Returns `nil` if the function is not successfully unregistered.

Example

```
viaUnregisterPreViaEngineCallback()
```

Mark Net Functions

The Mark Net command lets you visually trace the physical connectivity of a net in a layout design without having to use a schematic. You can use SKILL functions to work with the Mark Net command. For example, `leMarkNetGetNumThreads` returns the thread mode of the Mark Net command, and `leHiSaveAllHighLightMarkNet` opens the Save All Mark Nets form.

Related Topics

leHiMarkNet

leMarkNetGetNumThreads

leHiSaveAllHighLightMarkNet

leHiUnmarkNet

leHiUnmarkNetAll

leHiMarkNet

```
leHiMarkNet(
     [ w_windowId ]
     )
```

Description

Starts the *Connectivity – Nets – Mark* command and opens the MarkNet Options form in *w_windowId*, which lets you mark a net which you are prompted to select. If *w_windowId* is not specified, the current window is used.

After entering the SKILL code, select the net or nets you want to highlight.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

None

leHiSaveAllHighLightMarkNet

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

Description

Opens the *Save All Mark Nets* form.

Before running this SKILL function, use the *Mark Net* command (*Connectivity - Nets - Mark*) to highlight a net . In the *Save All Mark Nets* form, you can specify the cellview in which you want to save the highlighted nets.

Arguments

None

Values Returned

`t`	Returned when the form opens successfully.
`nil`	Returned when the functions fails to run.

Example

```
leHiSaveAllHighLightMarkNet()
```

leHiUnmarkNet

```
leHiUnmarkNet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Opens the Unmark Net form. If *w_windowId* is not specified, the current window is used.

After entering the SKILL code, select the net or nets whose highlighting you want to delete, or click the *Unmark All* button.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leHiUnmarkNetAll

```
leHiUnmarkNetAll(
     [ w_windowId ]
     )
     => t / nil
```

Description

Deletes the highlighting of all marked nets in *w_windowId*. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

leIsAnyMarkNetHighLighted

```
leIsAnyMarkNetHighLighted(
     w_windowId
     )
     => t / nil
```

Description

Checks if the specified window has a marked net.

Arguments

w_windowId	Window ID of the window specified.

Value Returned

t	The function is successful.
nil	The function is not successful.

Examples

```
winId    = hiGetCurrentWindow()
pt       = '(1 5)
netName  = "net1"
```

Marks a net from a shape at the specified point, `pt`:

```
leMarkNet(pt)
leIsAnyMarkNetHighLighted() => t
leUnmarkNet(pt) => t
leIsAnyMarkNetHighLighted() => nil
```

Marks a new trace using the net name `net1`:

```
leMarkNet(netName)
leIsAnyMarkNetHighLighted() => t
```

Removes all traces:

```
leUnmarkNet() => t
leIsAnyMarkNetHighLighted() => nil
```

leMarkNet

```
leMarkNet(
     l_point
     [ ?startLevel startLevel ]
     [ ?stopLevel stopLevel ]
     [ ?highlightIncrementally highlightIncrementally ]
     [ ?saveViaInfo saveViaInfo ]
     [ ?thickLine thickLine ]
     [ ?optionFile optionFile ]
     [ ?startLpp startLpp ]
     [ ?markNetColor markNetColor ]
     )
     => t / nil
```

Description

Marks a net in the current cellview. It traces the physical connectivity of a net(s) in a layout design starting from the input point (*l_point*) and highlights the traced net(s).

Arguments

l_point	The tracing will start from this point.
?startLevel *startLevel*	In a hierarchical design, tracing with start at this level.
?stopLevel *stopLevel*	In a hierarchical design, tracing with stop at this level.
?highlightIncrementally *highlightIncrementally*	The traced net is highlighted incrementally in case of value `t`. Valid values are `t` or `nil`.
?saveViaInfo *saveViaInfo*	While tracing, via information is saved in case of value `t`. Valid values are `t` or `nil`.
?thickLine *thickLine*	The traced net are highlighted in bold in case of value `t`. Valid values are `t` or `nil`.
?optionFile *optionFile*	Different rules like via layers rules, equivalent layers rules, and stop layer rules are read from the file.
?startLpp *startLpp*	The LPP from which tracing starts.
?markNetColor *markNetColor*	The marknet color. Valid values are `Y0`, `Y1`, `Y2`, `Y3`, `Y4`, `Y5`, `Y6`, `Y7`, `Y8`, or `Y9`.

Value Returned

`t`	Tracing has been performed successfully.
`nil`	Tracing has not been performed successfully.

Examples

```
leMarkNet('(5.23 3.6))
leMarkNet('(4.5 6.7) ?stopLevel 10)
```

```
leMarkNet('(4.5 6.7) ?saveViaInfo t)
leMarkNet('(4.5 6.7) ?optionFile "./newRuleFile")
leMarkNet('(4.5 6.7) ?startLevl 0 ?stopLevel 5 ?optionFile "./newRuleFile"
?saveViaInfo nil ?thickLine t)
leMarkNet(list(1.3 2.2) ?startLpp list("Metal1" "drawing"))
```

leMarkNetGetNumThreads

```
leMarkNetGetNumThreads(
     [ w_windowId ]
     )
     => integer
```

Description

Returns the thread mode of the Mark Net command. In Layout L, the Mark Net command always runs in a mono thread mode. In Layout XL and higher tiers, this command can run in auto or customizable thread mode.

Arguments

w_windowId	Window ID of the window specified.

Values Returned

`1`	Mark Net command will run in mono thread mode.
`< = 0`	Mark Net command will run in auto thread mode.
`> 1`	Mark Net command will run in customizable thread mode.

Example

```
leMarkNetGetNumThreads()
```

or

```
leMarkNetGetNumThreads(hiGetCurrentWindow())
```

leSaveMarkNet

```
leSaveMarkNet(
     cellName
     [ ?libName t_libName ]
     [ ?viewName t_viewName ]
     [ ?pt l_pt ]
     )
     => t / nil
```

Description

Saves a particular highlighted net at the specified point (`l_point`). If no point is specified, all the highlighted nets in current cellview are saved in the specified cellview (`lib`/`cell`/`view`).

Arguments

cellName	Specifies the name of the cellview in which the highlighted nets shapes are saved.
?libName *t_libName*	Specifies the name of the library of the cellview. The default value is the current cellview's library name in which the highlighted marknet is present.
?viewName *t_viewName*	Specifies the view name of the cellview. The default value is the current cellview's view name.
?pt *l_pt*	The mark net at this point will be saved. The default value is `nil`.

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Examples

```
leSaveMarkNet("save1")
```

```
or
leSaveMarkNet("save2" ?pt list(1 2))
or
leSaveMarkNet("save2" ?libName "gpdk090" ?pt list(1 2))
or
leSaveMarkNet("save2" ?libName "gpdk090" ?viewName "layout" ?pt list(1 2))
or
leSaveMarkNet("save3" ?libName "gpdk090" ?viewName "layout")
```

leUnmarkNet

```
leUnmarkNet(
     [ l_point ]
     => t / nil
```

Description

Deletes the highlighting of the marked nets in the current cellview. This function can delete the highlights of a particular net at the specified point (`l_point`). If no point is specified, the highlights of all the marked nets in current cellview are deleted.

Arguments

l_point	The highlights of the marked net that contains this point will be deleted.

Value Returned

`t`	The highlights of the marked nets have been deleted successfully.
`nil`	The highlight of the marked nets have not been deleted successfully.

Examples

```
leUnmarkNet('(5.23 3.6))
or
leUnmarkNet()
```

Palette Assistant Functions

Using the Palette assistant, you can define the window display context by setting the visibility and selectability of layers, objects, and grid items in a layout window. You can use SKILL functions to manage the Palette assistant. For example, you can use `pteContextMenu` to display the contextual menu for the specified Palette panel. You can use `pteEditLayerSetValidity` to edit the validity and membership state of layer-purpose pairs in selected layer sets.

Related Topics

leGetWirebondProfileVisible

leSetWirebondProfileVisible

pteCancelForm

pteClearSearchHistory

pteCloseLayerSetEdition

leGetWirebondProfileVisible

```
leGetWirebondProfileVisible(
     t_wpName
     )
     => t / nil
```

Description

(Virtuoso RF Option) Returns the visibility setting of the specified wirebond profile.

Arguments

t_wpName	Name of the wirebond profile.

Value Returned

`t`	The visibility setting of the wirebond profile is successfully returned.
`nil`	The visibility setting of the wirebond profile is not returned.

Example

Indicates that the `PROFILE1` wirebond profile is visible in the *Objects* panel of the palette.

```
leGetWirebondProfileVisible("PROFILE1")
t
```

leSetWirebondProfileVisible

```
leSetWirebondProfileVisible(
     t_wpName
     g_isVisible
     )
     => t / nil
```

Description

(Virtuoso RF Option) Sets the visibility of the specified wirebond profile.

Arguments

t_wpName	Name of the wirebond profile.
g_isVisible	The visibility setting of the wirebond profile. Valid values: `t`, `nil`

Value Returned

`t`	The visibility setting of the wirebond profile is successfully set.
`nil`	The visibility setting of the wirebond profile is not modified.

Example

Sets the visibility of `PROFILE1` wirebond profile to `t`.

```
leSetWirebondProfileVisible("PROFILE1" t)
t
```

pteCancelForm

```
pteCancelForm(
     t_formName
     [ w_windowId ]
     )
     => t / nil
```

Description

Closes the specified Palette form.

Arguments

t_formName	Name of the Palette form.
w_windowId	ID of the window in which the form is open. Valid values: ID of any open window Default: Current window

Value Returned

t	The specified Palette form was closed.
nil	The command was unsuccessful.

Examples

Closes the Options form:

```
pteCancelForm("Options")
```

Closes the Save As Layer Set form in the window with ID 2:

```
pteCancelForm("Save As Layer Set" window(2))
```

pteClearSearchHistory

```
pteClearSearchHistory(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Clears the search history for the specified Palette panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette for which you want to clear the search history. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The search history for the specified panel was cleared.
`nil`	The command was unsuccessful.

Examples

Clears the search history for the *Layers* panel in the current window:

```
pteClearSearchHistory("Layers")
```

Clears the search history for the *Grids* panel in the specified window:

```
pteClearSearchHistory("Grids" window(2))
```

pteCloseLayerSetEdition

```
pteCloseLayerSetEdition(
     [ w_windowId ]
     )
     => t / nil
```

Description

Closes layer set editing mode for the specified window, or for the current window if no window ID is specified.

Arguments

w_windowId	ID of the window containing the Palette for which you want to close the layer set editing mode. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Layer set editing mode was closed.
`nil`	The command was unsuccessful.

Examples

```
pteCloseLayerSetEdition()pteCloseLayerSetEdition(window(2))
```

pteCloseMPTSupportMode

```
pteCloseMPTSupportMode(
     [ w_windowId ]
     )
     => t / nil
```

Description

Hides the MPT-support-related Palette UI options.

Arguments

w_windowId	ID of the window in which you want to hide the MPT options. Valid values: ID of any open window Default: Current window

Value Returned

t	The options were hidden.
nil	The command was unsuccessful.

Examples

```
pteCloseMPTSupportMode()
pteCloseMPTSupportMode(window(2))
```

pteCollapse

```
pteCollapse(
     t_rowPath
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Collapses the specified item in the specified Palette panel in the specified window.

Arguments

t_rowPath	Path to the required item.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette in which you want to collapse an item. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified item was collapsed.
`nil`	The command was unsuccessful.

Examples

```
pteCollapse("Blockages" "Objects")
pteCollapse("Grids;Routing Grid" "Grids")
pteCollapse("Grids;Routing Grid" "Grids" window(2))
```

pteCollapseAll

```
pteCollapseAll(
     [ t_panelName ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Collapses all items in the specified Palette panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: Currently active panel; if no panel is active, all items in the *Layers* panel are collapsed
w_windowId	ID of the window containing the Palette in which you want to collapse items. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All items in the specified panel were collapsed.
`nil`	The command was unsuccessful.

Examples

```
pteCollapseAll("Grids")
pteCollapseAll("Objects" window(2))
```

pteContextMenu

```
pteContextMenu(
     [ t_panelName ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays the contextual menu for the specified Palette panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: Currently active panel; if no panel is active, the contextual menu for the *Layers* panel is displayed
w_windowId	ID of the window containing the Palette for which you want to display the contextual menu. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Contextual menu was displayed for the specified Palette panel.
`nil`	The command was unsuccessful.

Examples

```
pteContextMenu("Grids")
pteContextMenu("Objects" window(2))
```

pteDeleteLayerSet

```
pteDeleteLayerSet(
     t_layerSetName
     t_layerSetRepositoryPath
     )
     => t / nil
```

Description

Deletes the specified layer set and the corresponding layer set file.

Arguments

t_layerSetName	Name of the layer set.
t_layerSetRepositoryPath	Path to the layer set repository.

Value Returned

t	The layer set and the corresponding layer set file were deleted.
nil	The command was unsuccessful.

Examples

```
pteDeleteLayerSet("metal" ".cadence")
pteDeleteLayerSet("metal" "/share/.cadence")
```

pteDeselect

```
pteDeselect(
     t_rowPath
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Deselects an individual item in the specified Palette panel.

Arguments

t_rowPath	Path to the required item.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette in which you want to deselect an item. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified item was deselected.
`nil`	The command was unsuccessful.

Examples

```
pteDeselect("Nwell drawing" "Layers")
pteDeselect("Boundaries;Area Boundary" "Objects")
pteDeselect("Boundaries;Area Boundary" "Objects")
pteDeselect("Grids;Routing Grid;Metal1" "Grids" window(2))
```

pteDeselectAll

```
pteDeselectAll(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Deselects all items in the specified Palette panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette in which you want to deselect items. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All items in the specified Palette panel were deselected.
`nil`	The command was unsuccessful.

Examples

```
pteDeselectAll("Grids")
pteDeselectAll("Layers" window(2))
```

pteDiscardLayerSetEdition

```
pteDiscardLayerSetEdition(
     [ w_windowId ]
     )
     => t / nil
```

Description

Discards the changes made to the validity or membership state of a layer-purpose pair during the current layer set editing session.

Arguments

w_windowId	ID of the window containing the Palette in which you want to discard the changes. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The changes made during the current layer set editing session were discarded.
`nil`	The command was unsuccessful.

Examples

```
pteDiscardLayerSetEdition()
pteDiscardLayerSetEdition(window(2))
```

pteDockWindow

```
pteDockWindow(
     t_assistantName
     [ w_windowId ]
     )
     => t / nil
```

Description

Docks the specified Palette assistant.

Arguments

t_assistantName	Name of the assistant to be docked. Valid values: `Palette` in `SingleAssistant` mode; `Layers`, `Objects`, and `Grids` in `MultiAssistant` mode
w_windowId	ID of the window containing the Palette or Palette panel that you want to dock. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified Palette assistant was docked.
`nil`	The command was unsuccessful.

Examples

```
pteDockWindow("Layers")
pteDockWindow("Layers" window(2))
```

pteEditLayerSet

```
pteEditLayerSet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Enables addition or deletion of layer-purpose pairs in a layer set.

Arguments

w_windowId	ID of the window containing the Palette that you want to modify. Valid values: ID of any open window Default: Current window

Value Returned

t	The layer-purpose pair was added to or deleted from the layer set.
nil	The command was unsuccessful.

Examples

```
pteEditLayerSet()
pteEditLayerSet(window(2))
```

pteEditLayerSetValidity

```
pteEditLayerSetValidity(
     [ w_windowId ]
     )
     => t / nil
```

Description

Enables you to edit the validity and membership state of layer-purpose pairs in the selected layer sets.

Arguments

w_windowId	ID of the window containing the Palette that you want to modify. Valid values: ID of any open window Default: Current window

Value Returned

t	The layer-purpose pair validity state is set to editable.
nil	The command was unsuccessful.

Examples

```
pteEditLayerSetValidity()
pteEditLayerSetValidity(window(2))
```

pteExpand

```
pteExpand(
     t_rowPath
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Expands the specified item in the specified Palette panel in the specified window.

Arguments

t_rowPath	Path to the item.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette in which you want to expand an item. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified row was expanded.
`nil`	The command was unsuccessful.

Examples

```
pteExpand("Blockages" "Objects")
pteExpand("Grids;Routing Grid" "Grids")
pteExpand("Grids;Routing Grid" "Grids" window(2))
```

pteExpandAll

```
pteExpandAll(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Expands all objects in the specified Palette panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: Currently active panel; if no panel is active, all items in the *Layers* panel are expanded
w_windowId	ID of the window containing the Palette in which you want to expand items. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All items in the specified panel were expanded.
`nil`	The command was unsuccessful.

Examples

```
pteExpandAll("Grids")
pteExpandAll("Objects" window(2))
```

pteExportLayerSet

```
pteExportLayerSet(
     t_layerSetName
     t_layerSetRepositoryPath
     t_layerSetFilePath
     [ g_overwrite ]
     )
     => t / nil
```

Description

Exports a layer set from a Palette layer set repository to the file system.

Arguments

t_layerSetName	Name of the layer set to be exported.
t_layerSetRepositoryPath	The path to the layer set repository that contains the layer set to be exported.
t_layerSetFilePath	The path to the location where you want to save the exported layer set. Include in this path the filename (*filename*.layerSet) with which you want to save the exported layer set.
g_overwrite	If set to t, any existing layer set file with the same name is overwritten. Default value: t

Value Returned

t	The specified layer set was exported.
nil	The command was unsuccessful.

Examples

```
pteExportLayerSet("metal" ".cadence" "/tmp/metal.layerSet")
pteExportLayerSet("metal" "/share/.cadence" "/tmp/metal.layerSet")
pteExportLayerSet("metal" ".cadence" "/tmp/metal.layerSet" t)
```

pteFindNext

```
pteFindNext(
     [ t_panelName ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Finds and sets as active the next layer-purpose pair that matches the search string, in the specified panel. This function works only if the *Search Mode* option in the Palette Options form is set to *Filter*.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: Currently active panel; if no panel is active, the search is performed in the `Layers` panel
w_windowId	ID of the window containing the Palette that you want to modify. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The next layer-purpose pair that matches the search string was found and set as active.
`nil`	The command was unsuccessful.

Examples

```
pteFindNext("Layers")
pteFindNext("Layers" window(2))
```

pteFindPrev

```
pteFindPrev(
     [ t_panelName ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Finds and sets as active the previous layer-purpose pair that matches the search string, in the specified panel. This function works only if the *Search Mode* option in the Palette Options form is set as *Filter*.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: Currently active panel; if no panel is active, the search is performed in the `Layers` panel
w_windowId	ID of the window containing the Palette that you want to modify. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The previous layer-purpose pair that matches the search string was found and set as active.
`nil`	The command was unsuccessful.

Examples

```
pteFindPrev("Layers")
pteFindPrev("Layers" window(2))
```

pteGetActiveScopeModes

```
pteGetActiveScopeModes(
     [ w_windowId ]
     )
     => l_result / nil
```

Description

Returns a list containing the scope modes that are set as active in the specified window.

Arguments

w_windowId	ID of the window containing the Palette from which you want to retrieve scope modes data. Valid values: ID of any open window Default: Current window

Value Returned

l_result	A SKILL list of scope modes that are set in the window.
nil	The command was unsuccessful.

Examples

If the active scopes in the current window are Routing and Valid, `pteGetActiveScopeModes()` returns the list `("Routing" "Valid")`.

```
pteGetActiveScopeModes()
pteGetActiveScopeModes(window(2))
```

pteGetAllActiveLayerSets

```
pteGetAllActiveLayerSets(
     [ w_windowId ]
     )
     => l_layerSets / nil
```

Description

Returns a list containing the layer sets that are set as active in the design open in the specified window.

Arguments

w_windowId	ID of the window containing the Palette from which you want to retrieve layer set data. Valid values: ID of any open window Default: Current window

Value Returned

l_layerSets	A layer set or a list of layer sets.
nil	The command was unsuccessful.

Examples

```
pteGetAllActiveLayerSets()
pteGetAllActiveLayerSets(window(2))
```

pteGetAllLayerSets

```
pteGetAllLayerSets(
     [ w_windowId ]
     )
     => l_layerSets / nil
```

Description

Returns a list containing all the layer sets available in the design open in the specified window.

Arguments

w_windowId	ID of the window containing the Palette from which you want to retrieve layer set data. Valid values: ID of any open window Default: Current window

Value Returned

l_layerSets	A layer set or a list of layer sets.
nil	The command was unsuccessful.

Examples

```
pteGetAllLayerSets()
pteGetAllLayerSets(window(2))
```

pteGetCurrentPanel

```
pteGetCurrentPanel(
     )
     => t_panelName / nil
```

Description

Retrieves the name of the currently active Palette panel.

Arguments

None

Value Returned

t_panelName	Name of the currently active Palette panel.
nil	The command was unsuccessful.

Examples

```
pteGetCurrentPanel()
pteGetCurrentPanel(window(2))
```

pteGetLayerPath

```
pteGetLayerPath(
     t_layerPurposeName
     [ w_windowId ]
     )
     => t_elementPath / nil
```

Description

Retrieves the element path of the specified layer-purpose pair when the *View By* mode is applied to the Palette. The result can be used to provide the *t_paletteElementPath* argument needed by the pteSetVisible, pteSetSelectability, pteSetValidity, pteSetOnlyVisible, pteSetOnlySelectable, ptePropagateVisibility, and ptePropagateSelectability functions in the *View By* mode.

Arguments

t_layerPurposeName	Name of the layer-purpose pair for which the element path needs to be displayed.
w_windowId	ID of the window containing the Palette from which you want to retrieve data. Valid values: ID of any open window Default: Current window

Value Returned

t_elementPath	The element path of the specified layer-purpose pair.
nil	The command was unsuccessful.

Example

Uses the output of the pteGetLayerPath function as an input for the pteSetVisible function.

```
pteGetLayerPath("Oxide drawing")
pteSetVisible(pteGetLayerPath("Oxide drawing"))
```

pteGetLayerSetColoredLpp

```
pteGetLayerSetColoredLpp(
     t_layerSetName
     [ w_windowId ]
     )
     => t_layerPurposeName / nil
```

Description

Returns the layer-purpose pair associated with the specified layer set.

Arguments

t_layerSetName	Name of the layer set.
w_windowId	ID of the window containing the Palette from which you want to retrieve data. Valid values: ID of any open window Default: Current window

Value Returned

t_layerPurposeName	Name of the layer-purpose pair associated with the specified layer set.
nil	The command was unsuccessful.

Example

```
pteGetLayerSetColoredLpp("Poly")
```

pteGetLayerSetMembers

```
pteGetLayerSetMembers(
     t_layerSetName
     [ w_windowId ]
     )
     => l_result / nil
```

Description

Returns the attribute settings for layer-purpose pairs, objects, and grids that are members of the specified layer set. For layer-purpose pairs, attributes include visibility, selectability, validity, priority, and routing direction (if set). For objects, attributes include visibility and selectability; and for grids, the attribute returned is visibility.

Arguments

t_layerSetName	Name of the layer set.
w_windowId	ID of the window containing the Palette from which you want to retrieve data. Valid values: ID of any open window Default: Current window

Value Returned

l_result	A list of layer set members along with the attribute values.
nil	The command was unsuccessful.

Example

```
pteGetLayerSetMembers("All Layers")
```

pteGetLPPDisplayedInPalette

```
pteGetLPPDisplayedInPalette(
     [ w_windowId ]
     )
     => t_layerPurposeNames / nil
```

Description

Lists the layer-purpose pairs currently displayed in the *Layers* panel.

Arguments

w_windowId	ID of the window containing the Palette from which you want to retrieve the layer-purpose pair names. Valid values: ID of any open window Default: Current window

Value Returned

t_layerPurposeNames	List of layer-purpose pairs currently displayed in the *Layers* panel.
`nil`	The command was unsuccessful.

Examples

```
pteGetLPPDisplayedInPalette()
```

Generates the following output, assuming that the layer-purpose pairs have been filtered to display only those that contain the string 'RX':

```
    (("RX" "drawing")
         ("RX" "net")
         ("RX" "pin")
         ("RX" "label")
         ("RX" "ing")
    )
pteGetLPPDisplayedInPalette(window(2))
```

pteGetLSAtPosition

```
pteGetLSAtPosition(
     x_position
     [ w_windowId ]
     )
     => t_layersetName / nil
```

Description

Displays the name of the layer set that is listed at the specified position in the *Layer Set Manager*.

Arguments

x_position	Position at which the layer set is listed in the *Layer Set Manager*.
w_windowId	ID of the window containing the Palette from which you want to retrieve the layer set name. Valid values: ID of any open window Default: Current window

Value Returned

t_layersetName	Name of the layer set that is located at the specified position was displayed.
nil	The command was unsuccessful.

Examples

```
pteGetLSAtPosition(0)
pteGetLSAtPosition(1 window(2))
```

pteGetMPTSupportMode

```
pteGetMPTSupportMode(
     w_windowId
     )
     => t / nil
```

Description

Determines if MPT support is enabled in the window with the specified ID.

Arguments

w_windowId	ID of any open window.

Value Returned

t	MPT support is enabled in the window with the specified ID.
nil	MPT support is disabled in the window with the specified ID.

Examples

```
pteGetMPTSupportMode(hiGetCurrentWindow())
pteGetMPTSupportMode(window(3))
```

pteGetNextLayerSets

```
pteGetNextLayerSets(
     [ w_windowId ]
     )
     => l_layerSets / nil
```

Description

Returns the layer set listed immediately below the layer set if only one layer set is selected in the *Layer Set Manager*. If multiple layer sets are selected in the *Layer Set Manager*, returns a list comprising layer sets listed immediately below each selected layer set.

Arguments

w_windowId	ID of the window containing the Palette from which you want to retrieve layer set data. Valid values: ID of any open window Default: Current window

Value Returned

l_layerSets	A layer set or a list of layer sets.
nil	The selected layer set is the last layer set listed in the *Layer Set Manager*.

Examples

```
pteGetNextLayerSets()
pteGetNextLayerSets(window(2))
```

pteGetPrevLayerSets

```
pteGetPrevLayerSets(
     [ w_windowId ]
     )
     => l_layerSets / nil
```

Description

Returns the layer set listed immediately above the layer set if only one layer set is selected in the *Layer Set Manager*. If multiple layer sets are selected in the *Layer Set Manager*, returns a list comprising layer sets listed immediately above each selected layer set.

Arguments

w_windowId	ID of the window containing the Palette from which you want to retrieve layer set data. Valid values: ID of any open window Default: Current window

Value Returned

l_layerSets	A layer set or a list of layer sets.
nil	The selected layer set is the first layer set listed in the *Layer Set Manager*.

Examples

```
pteGetPrevLayerSets()
pteGetPrevLayerSets(window(2))
```

pteGetSelection

```
pteGetSelection(
     t_panelName
     [ w_windowId ]
     )
     => l_layerSetItems / nil
```

Description

Retrieves a list of the selected layer-purpose pairs, objects, or grids from the specified panel.

Arguments

t_panelName	Name of the panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette from which you want to retrieve a list of selected objects. Valid values: ID of any open window Default: Current window

Value Returned

l_layerSetItems	A list containing the selected layer-purpose pairs, objects, or grids.
`nil`	If the panel specified is *Objects* or *Grids*, but no object or grid is selected in the panel. If the panel specified is *Layers* and the *Selection Mode* is set to *Single Selection* (the default value), this function returns `nil` even though a layer-purpose pair is selected (the active layer-purpose pair). **Note:** If *Selection Mode* is set to *Single Selection*, use leGetEntryLayer to retrieve the active layer-purpose pair.

Examples

```
pteGetSelection("Layers")
(("Oxide drawing" "Layers") ("Poly drawing" "Layers"))
pteGetSelection("Grids" window(2))
```

```
(("Grids;Routing Grid;Metal1" "Grids")
    ("Grids;Routing Grid;Metal2" "Grids"))
```

pteHideAllTools

```
pteHideAllTools(
     [ w_windowId ]
     )
     => t / nil
```

Description

Hides all Palette tools and toolbars.

Arguments

w_windowId	ID of the window containing the Palette that you want to modify. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All Palette tools and toolbars were hidden.
`nil`	The command was unsuccessful.

Examples

```
pteHideAllTools()
pteHideAllTools(window(2))
```

pteImportLayerSet

```
pteImportLayerSet(
     t_layerSetFilePath
     t_layerSetRepositoryPath
     t_layerSetName
     [ g_overwrite ]
     )
     => t / nil
```

Description

Imports the specified layer set file from the file system to the Palette repository and saves it with the specified name.

Arguments

t_layerSetFilePath

Path to the layer set file that is to be imported.

t_layerSetRepositoryPath

Path to the layer set repository.

t_layerSetName	The name with which the imported layer set file needs to be saved in the layer set repository.
g_overwrite	If set to `t`, replaces an existing layer set file if it has the same name as that specified for the file being imported.

Value Returned

`t`	The specified layer set was imported.
`nil`	The command was unsuccessful.

Examples

```
pteImportLayerSet("/tmp/metal.layerSet" ".cadence" "metal")
pteImportLayerSet("/tmp/metal.layerSet" "/share/.cadence" "metal" t)
```

pteInfraGetPaletteMode

```
pteInfraGetPaletteMode(
     )
     => SingleAssistant / MultiAssistant / SingleWindow
```

Description

Returns the container type of the Palette that is currently active.

Arguments

None

Value Returned

`SingleAssistant`	Palette container is set to Single Assistant.
`MultiAssistant`	Palette container is set to Multiple Assistant.
`SingleWindow`	Palette container is set to Single Window.

Example

```
pteInfraGetPaletteMode()
```

pteIsSelectable

```
pteIsSelectable(
     { t_layerPurposeName | t_paletteElementPath }
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Retrieves the selectability state for the specified layer-purpose pair or object.

Arguments

t_layerPurposeName

Layer-purpose pair or object name.

t_paletteElementPath

Path to the layer-purpose pair or object for which you want to retrieve the selectability state.

t_panelName	Name of the Palette panel. Valid values: `Layers` and `Objects`
w_windowId	ID of the window that contains the Palette from which you want to retrieve data. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer-purpose pair or object is selectable.
`nil`	The specified layer-purpose pair or object is not selectable.

Examples

- When *View By* is set to *None*:

```
pteIsSelectable("prBoundary boundary" "Layers")
pteIsSelectable("prBoundary boundary" "Layers" window(2))
```

- When *View By* is set to *Layer*:

```
pteIsSelectable("prBoundary;prBoundary boundary" "Layers")
```

- When *View By* is set to *Purpose*:

```
pteIsSelectable("boundary;prBoundary boundary" "Layers")
```

Related Topics

pteSetSelectable

pteIsVisible

```
pteIsVisible(
     { t_layerPurposeName | t_paletteElementPath }
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Retrieves the visibility state for the specified layer-purpose pair, object, or grid.

Arguments

t_layerPurposeName

Layer-purpose pair, object, or grid name.

t_paletteElementPath

Path to the layer-purpose pair, object, or grid for which you want to retrieve the visibility state.

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window that contains the Palette from which you want to retrieve data. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer-purpose pair, object, or grid is set to visible.
`nil`	The specified layer-purpose pair, object, or grid is not set to visible.

Examples

- When *View By* is set to *None*:

```
pteIsVisible("prBoundary boundary" "Layers")
pteIsVisible("prBoundary boundary" "Layers" window(2))
```

- When *View By* is set to *Layer*:

```
pteIsVisible("prBoundary;prBoundary boundary" "Layers")
```

- When *View By* is set to *Purpose*:

```
pteIsVisible("boundary;prBoundary boundary" "Layers")
```

Related Topics

pteSetVisible

pteLoadConfig

```
pteLoadConfig(
     t_configFilePath
     [ w_windowId ]
     )
     => t / nil
```

Description

Loads the specified Palette configuration file. A Palette configuration file contains Palette-related information, such as the columns and toolbars to be displayed and their location.

Arguments

t_configFilePath	Path to the Palette configuration file that is to be loaded.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The Palette configuration file was loaded.
`nil`	The command was unsuccessful.

Examples

```
pteLoadConfig(".cadence/dfII/palette/palette.ini")
pteLoadConfig(".cadence/dfII/palette/palette.ini" window(2))
```

pteLoadDefaults

```
pteLoadDefaults(
     [ w_windowId ]
     )
     => t / nil
```

Description

Loads the display context with the default values. A display context determines display properties, such as visibility and selectability of layer-purpose pairs, objects, and grids.

Arguments

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The display context was loaded with the default values.
nil	The command was unsuccessful.

Examples

```
pteLoadDefaults(window(2))
```

pteLoadFromTechFile

```
pteLoadFromTechFile(
     [ w_windowId ]
     )
     => t / nil
```

Description

Loads the display context from the technology file into the Palette. A display context determines display properties such as visibility and selectability of layer-purpose pairs, objects, and grids.

Arguments

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The display context from the technology file was loaded into the Palette.
nil	The command was unsuccessful.

Examples

```
pteLoadFromTechFile()
pteLoadFromTechFile(window(2))
```

pteLoadGDSNumber

```
pteLoadGDSNumber(
     t_gdsNumberLayermapFilePath
     [ w_windowId ]
     )
     => t / nil
```

Description

Loads the GDS number layermap file from the specified location into the Palette.

Arguments

t_gdsNumberLayermapFilePath	Path to the GDS number layermap file that is to be loaded.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The GDS number layer map file was loaded.
nil	The command was unsuccessful.

Examples

```
pteLoadGDSNumber("./lib.layermap")
pteLoadGDSNumber("./lib.layermap" window(2))
```

pteLoadLayerSet

```
pteLoadLayerSet(
     t_layerSetFileName
     t_layerSetRepositoryPath
     )
     => t / nil
```

Description

Loads a layer set from a layer set file existing in a palette repository.

Arguments

t_layerSetFileName

Name of the layer set file.

t_layerSetRepositoryPath

Fully qualified path of a valid repository.

Value Returned

t	The layer set was loaded.
nil	he layer set was not loaded.

Examples

```
pteLoadLayerSet("metalset.layerSet" "./.cadence")
```

pteLoadLSWInfo

```
pteLoadLSWInfo(
     t_configFilePath
     [ w_windowId ]
     )
     => t / nil
```

Description

Loads the specified Layer Selection Window (LSW) information file into the Palette. The visibility and selectability settings stored in the LSW information file for layer-purpose pairs and objects are loaded into the Palette.

Arguments

t_configFilePath	Path to the LSW Info file that is to be loaded.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The LSW information file was loaded.
nil	The command was unsuccessful.

Examples

```
pteLoadLSWInfo("./palette.info")
pteLoadLSWInfo("./palette.lsw" window(2))
```

pteMapWindow

```
pteMapWindow(
     t_assistantName
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays, if hidden, the specified Palette panel.

Arguments

t_assistantName	Name of the panel to be mapped. Valid values: `Layers`, `Objects`, and `Grids` in `SingleAssistant` mode
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified Palette panel was displayed.
`nil`	The command was unsuccessful.

Examples

```
pteMapWindow("Layers")
pteMapWindow("Layers" window(2))
```

pteMinimizeSingleWindowPalette

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

Description

Minimizes the Palette, but only if the Palette container type is set to Single Window.

Arguments

None

Value Returned

`t`	The Palette was minimized.
`nil`	The command was unsuccessful.

Example

```
pteMinimizeSingleWindowPalette()
```

pteMoveLayerSelection

```
pteMoveLayerSelection(
     x_position
     [ w_windowId ]
     )
     => t / nil
```

Description

Moves the layer-purpose pairs selected in the *Layers* panel to the specified position. This function works only if the *View By* mode is set to *None*.

Arguments

x_position	Position to which the selected layer-purpose pairs need to be moved.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The selected layer-purpose pairs were moved to the specified location.
nil	The command was unsuccessful.

Examples

```
pteMoveLayerSelection(0)
pteMoveLayerSelection(100 window(2))
```

pteMoveSingleWindowPalette

```
pteMoveSingleWindowPalette(
     x_xcoordinate
     x_ycoordinate
     )
     => t / nil
```

Description

Places the upper-left corner of the Palette at the specified x- and y-coordinate, if the Palette container type is set to Single Window.

Arguments

x_xcoordinate	An integer value that represents the x-coordinate.
x_ycoordinate	An integer value that represents the y-coordinate.

Value Returned

`t`	The Palette was moved to the specified location.
`nil`	The command was unsuccessful.

Example

```
pteMoveSingleWindowPalette(50 100)
```

pteMPTSupportMode

```
pteMPTSupportMode(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays the MPT-support-related Palette UI options.

Arguments

w_windowId	ID of the window in which you want to display the MPT options. Valid values: ID of any open window Default: Current window

Value Returned

t	The MPT options were displayed.
nil	The command was unsuccessful.

Examples

```
pteMPTSupportMode()
pteMPTSupportMode(window(2))
```

pteOpenForm

```
pteOpenForm(
     t_formName
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays the specified Palette form.

Arguments

t_formName	Name of the Palette form.
w_windowId	ID of the window in which you want to display the form. Valid values: ID of any open window Default: Current window

Value Returned

t	The specified Palette form was displayed.
nil	The command was unsuccessful.

Examples

Opens the Palette Options form:

```
pteOpenForm("Options")
```

Opens the Save As Layer Set form in the window with ID 2:

```
pteOpenForm("Save As Layer Set" window(2))
```

ptePropagateSelectability

```
ptePropagateSelectability(
     { t_layerPurposeName | t_paletteElementPath }
     g_selectabilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Propagates selectability changes to child items when the parent and child items are in a desynchronized state. Specify either *t_layerPurposeName* or *t_paletteElementPath*.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_paletteElementPath

Path to the parent item.

g_selectabilityState

If set to `t`, selectability changes are propagated to child items (if the parent and child items are desynchronized).

t_panelName

Name of the Palette panel.
Valid values: `Layers` and `Objects`

w_windowId

ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	Selectability status was propagated to child items.
`nil`	The command was unsuccessful.

Examples

```
ptePropagateSelectability("Shapes" t "Objects")
```

ptePropagateVisibility

```
ptePropagateVisibility(
     { t_layerPurposeName | t_paletteElementPath }
     g_visibilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Propagates visibility changes to child items when the parent and child items are in a desynchronized state. Specify either *t_layerPurposeName* or *t_paletteElementPath*.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_paletteElementPath

Path to the parent item.

g_selectabilityState

If set to `t`, visibility changes are propagated to child items (if the parent and child items are desynchronized).

t_panelName — Name of the Palette panel.
Valid values: `Layers`, `Objects`, and `Grids`

w_windowId — ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	Visibility status was propagated to child items.
`nil`	The command was unsuccessful.

Examples

```
ptePropagateVisibility("Shapes" t "Objects")
ptePropagateVisibility("Grids;Routing Grid" t "Grids" window(2))
```

pteRaiseSingleWindowPalette

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

Description

Places the Palette on top of the other open windows, if the Palette container type is set to Single Window.

Arguments

None

Value Returned

`t`	Palette was placed on top of the other open windows.
`nil`	The command was unsuccessful.

Example

```
pteRaiseSingleWindowPalette()
```

pteRegisterUserLSManagerTrigger

```
pteRegisterUserLSManagerTrigger(
     t_funcName
     )
     => t / nil
```

Description

Registers a trigger, which is a user-defined SKILL procedure, from CIW. This trigger is called with an argument list containing *modifier*, *mouseButton*, *event*, *editedLS*, and *activeLSList* when an action (other than a drag-and-drop action) is performed in the *Layer Set Manager*.

Valid values of these arguments are as follows:

- *modifier*: `None`, `Shift`, and `Ctrl`
- *mouseButton*: `Left` and `Right`
- *event*: `Layer Set Activation`, `Visibility ON`, `Visibility OFF`, `Selectability ON`, `Selectability OFF`, `EnableLS`, and `DisableLS`
- *editedLS*: Layer set on which the user clicks
- *activeLSList*: List of all currently active layer sets

Only one trigger can be registered at a time; it can be unregistered using pteUnRegisterUserLSManagerTrigger.

Arguments

t_funcName	Name of the trigger to be registered.

Value Returned

`t`	The trigger was successfully registered.
`nil`	The command was unsuccessful.

Example

```
pteRegisterUserLSManagerTrigger("mytrigger")
```

The following is a sample implementation of `mytrigger`:

```
procedure( mytrigger(val)
    let(()
    info("%L" val->??)
    t
    )
    );
```

pteRegisterUserSelectionTrigger

```
pteRegisterUserSelectionTrigger(
     t_funcName
     )
     => t / nil
```

Description

Registers a trigger, which is a user-defined SKILL procedure, from CIW. This trigger is called with an argument list containing *windowId*, *layerName*, *purposeName*, *mouseButton*, and *modifier* for every selection action. Valid values for *mouseButton* are `Left`, `Right`, and `Middle`; valid values for *modifier* are `None`, `Shift`, and `Ctrl`.

Only one trigger can be registered at a time; it can be unregistered using pteUnRegisterUserSelectionTrigger.

Note: The trigger works only in *Single Selection* mode. In *Multi Selection* mode, the trigger is ignored.

Arguments

t_funcName	Name of the trigger to be registered.

Value Returned

`t`	The trigger was successfully registered.
`nil`	The command was unsuccessful.

Example

```
pteRegisterUserSelectionTrigger("mytrigger")
```

The following is a sample implementation of `mytrigger`:

```
procedure(mytrigger(windowId layerName layerPurpose button modifier)
    prog( ()
        if(layerName == "Metal1" &&
            layerPurpose == "net" &&
            button == "Left" &&
            modifier == "None"
        then
```

```
        pteSetSelectable("Metal1 drawing" nil "Layers")
    else
    if(layerName == "Metal1" &&
        layerPurpose == "net" &&
        button == "Left" &&
        modifier == "Shift"
    then
        pteSetSelectable("Metal1 drawing" t "Layers")
    else
    if(layerName == "Metal1" &&
        layerPurpose == "net" &&
        button == "Middle" &&
        modifier == "None"
    then
        pteSetSelectable("Metal1 slot" nil "Layers")
    else
    if(layerName == "Metal1" &&
        layerPurpose == "net" &&
        button == "Middle" &&
        modifier == "Shift"
    then
        pteSetSelectable("Metal1 slot" t "Layers")
    else
    if(layerName == "Metal1" &&
        layerPurpose == "net" &&
        button == "Right" &&
        modifier == "None"
    then
        pteSetSelectable("Metal1 pin" nil "Layers")
    else
    if(layerName == "Metal1" &&
        layerPurpose == "net" &&
        button == "Right" &&
        modifier == "Shift"
    then
        pteSetSelectable("Metal1 pin" t "Layers")
)
)
)
)
)
```

```
    )
    ); prog
); procedure
```

pteReloadLayerSet

```
pteReloadLayerSet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Reloads the layer set file for the layer set that is currently in use. This resynchronizes the visibility and selectability status of the layers, objects, or grids items with that available in the layer set file.

Arguments

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The layer set was reloaded.
nil	The command was unsuccessful.

Examples

```
pteReloadLayerSet()
pteReloadLayerSet(window(2))
```

pteResizeSingleWindowPalette

```
pteResizeSingleWindowPalette(
     x_width
     x_height
     )
     => t / nil
```

Description

Resizes the Palette by using the specified width and height values, if the Palette container type is set to Single Window.

Arguments

x_width	An integer value that represents the width.
x_height	An integer value that represents the height.

Value Returned

`t`	Palette was resized.
`nil`	The command was unsuccessful.

Example

```
pteResizeSingleWindowPalette(200 400)
```

pteSaveAsLayerSet

```
pteSaveAsLayerSet(
     t_layerSetName
     t_layerSetRepositoryPath
     [ g_overwrite ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the layer set currently in use with a new name.

Arguments

t_layerSetName	Name with which the layer set is to be saved.
t_layerSetRepositoryPath	Path to the layer set repository.
g_overwrite	If set to `t`, any existing layer set file with the same name is overwritten. Default value: `t`
w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer set was saved with the specified name.
`nil`	The command was unsuccessful.

Examples

```
pteSaveAsLayerSet("metal" ".cadence")
pteSaveAsLayerSet("metal" "/share/.cadence")
pteSaveAsLayerSet("metal" ".cadence" t)
pteSaveAsLayerSet("metal" ".cadence" t window(2))
```

pteSaveAsSynchronizedLayerSet

```
pteSaveAsSynchronizedLayerSet(
     t_layerSetName
     t_layerSetRepositoryPath
     [ g_overwrite ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the layer set currently in use, as a synchronized layer set, with the specified name in the specified layer set repository. The newly saved layer set is set as active.

For information about synchronized layer sets, see Creating a Layer Set.

Arguments

t_layerSetName	Name with which the layer set is to be saved.
t_layerSetRepositoryPath	Path to the layer set repository in which you want to save the layer set.
g_overwrite	If set to `t`, any existing layer set file with the same name is overwritten. Default value: `t`
w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer set was saved as a synchronized layer set in the specified layer set repository.
`nil`	The command was unsuccessful.

Examples

```
pteSaveAsSynchronizedLayerSet("meta2_syn" ".cadence")
```

Saves the layer set currently in use with the name `meta2_syn.layerSet` in the `./.cadence/dfII/layerSets/<`*`technology library name`*`>` repository.

```
pteSaveAsSynchronizedLayerSet("meta2_syn" "/home/user01/.cadence")
```

Saves the layer set currently in use with the name `meta2_syn.layerSet` in the `/home/user01/.cadence/dfII/layerSets/<`*`technology library name`*`>` repository.

pteSaveConfig

```
pteSaveConfig(
     t_configFilePath
     [ g_overwrite ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the Palette configuration information in the specified file. This includes information about the Palette settings, such as the columns and toolbars currently displayed in the Palette.

Arguments

t_configFilePath	Path to the location where you want to save the Palette configuration file.
g_overwrite	If set to `t`, any existing configuration file with the same name is overwritten.
w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The Palette configuration information was saved.
`nil`	The command was unsuccessful.

Examples

```
pteSaveConfig(".cadence/dfII/palette/palette.ini")
pteSaveConfig(".cadence/dfII/palette/palette.ini" t)
pteSaveConfig(".cadence/dfII/palette/palette.ini" t window(2))
```

pteSaveGDSNumber

```
pteSaveGDSNumber(
     g_overwrite
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the active GDSII number layermap file.

Arguments

g_overwrite	If set to `t`, any existing GDSII number layermap file is overwritten.
w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The GDSII number layermap file was saved.
`nil`	The command was unsuccessful.

Examples

```
pteSaveGDSNumber(t)
pteSaveGDSNumber(t window(2))
```

pteSaveLayerSet

```
pteSaveLayerSet(
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the changes made to the layer set that is currently in use.

Arguments

w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

t	The specified layer set was saved.
nil	The command was unsuccessful.

Examples

```
pteSaveLayerSet()
pteSaveLayerSet(window(2))
```

pteSaveLayerSetList

```
pteSaveLayerSetList(
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the list of layer sets displayed in the *Layer Set Manager* to a file named `layerset.order`. The order in which the layer sets are listed and the enable state of each layer set is also saved.

Arguments

w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer sets listed in *Layer Set manager* were saved.
`nil`	The command was unsuccessful.

Examples

```
pteSaveLayerSetList()
pteSaveLayerSetList(window(2))
```

pteSaveLayerSetListInRepository

```
pteSaveLayerSetListInRepository(
     t_layerSetRepositoryPath
     [ g_overwrite ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the list of layer sets displayed in the *Layer Set Manager* to a file named `layerset.order` in the specified repository. The order in which the layer sets are listed and their state—whether enabled or disabled—is also saved.

Arguments

t_layerSetRepositoryPath

Path to which you want to save the `layerset.order` file.

g_overwrite

If set to `t`, the existing `layerset.order` file is overwritten.
Default value: `t`

w_windowId

ID of the window containing the Palette from which you want to save information.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`

The `layerset.order` file was successfully saved at the specified location.

`nil`

The command was unsuccessful.

Examples

Saves the `layerset.order` file to the present working directory:

```
pteSaveLayerSetListInRepository("./.cadence")
```

Saves the `layerset.order` file to the user's home directory:

```
pteSaveLayerSetListInRepository("/home/user01/.cadence")
```

pteSaveLSWInfo

```
pteSaveLSWInfo(
     t_configFilePath
     [ g_overwrite ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the current LSW Info file at the specified location. This file contains information about the state of the Palette. It does not store the color and lock information of layer-purpose pairs.

Arguments

t_configFilePath	Path to the location where you want to save the LSW Info file.
g_overwrite	If set to `t`, any existing LSW Info file with the same name is overwritten.
w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The LSW Info file was saved.
`nil`	The command was unsuccessful.

Examples

```
pteSaveLSWInfo("./palette.info")
pteSaveLSWInfo("./palette.lsw" t)
pteSaveLSWInfo("./palette.file" t window(2))
```

pteSaveToTechFile

```
pteSaveToTechFile(
     [ w_windowId ]
     )
     => t / nil
```

Description

Saves the display context from the Palette to the technology file. A display context determines display properties such as visibility and selectability of layer-purpose pairs, objects, and grids. The display context stored in individual layer sets is not considered.

Arguments

w_windowId	ID of the window containing the Palette from which you want to save information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The information from the Palette was saved to the technology file.
`nil`	The command was unsuccessful.

Examples

```
pteSaveToTechFile()
pteSaveToTechFile(window(2))
```

pteSelect

```
pteSelect(
     t_rowPath
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Selects the specified layer, object, or grid. An item selected previously is not deselected when you run the command again to select another item. This function works only if the *Selection Mode* is set to *Multi Selection*.

Arguments

t_rowPath	Path to the layer, object, or grid that you want to select.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette in which you want to select an item. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer, object, or grid was selected.
`nil`	The command was unsuccessful.

Examples

```
pteSelect("Nwell drawing" "Layers")
pteSelect("Boundaries;Area Boundary" "Objects")
pteSelect("Boundaries;Area Boundary" "Objects")
pteSelect("Grids;Routing Grid;Metal1" "Grids" window(2))
```

pteSelectAll

```
pteSelectAll(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Selects all items in the specified Palette panel. This function works only if the *Selection Mode* is set to *Multi Selection*.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette in which you want to select the items. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All items in the specified panel were selected.
`nil`	The command was unsuccessful.

Examples

```
pteSelectAll("Grids")
pteSelectAll("Layers" window(2))
```

pteSetActiveLpp

```
pteSetActiveLpp(
     t_layerPurposeName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the specified layer-purpose pair as active. A layer that is not defined in the technology file is not displayed in the Palette, and an invalid layer cannot be set as active. Therefore, ensure that the specified layer is defined in the technology file and is valid.

Arguments

t_layerPurposeName	Name of the layer-purpose pair to be set as active.
w_windowId	ID of the window in which the layer-purpose pair is to be set as active. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer-purpose pair was set as active.
`nil`	The command was unsuccessful.

Examples

```
pteSetActiveLpp("nwell drawing")
pteSetActiveLpp("oxide drawing" window(2))
```

pteSetActiveLppColor

```
pteSetActiveLppColor(
     { grayColor | mask1Color | mask2Color }
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the specified color for the active layer-purpose pair. The lock state is set to unlocked.

The `pteSetActiveLppColor` SKILL function does not change the default color of an active layer. To change the layer default color, use mptSetLayerDefaultColor.

Arguments

`grayColor	mask1Color	mask2Color`	The color to be set for the active layer-purpose pair. Valid values: `grayColor`, `mask1Color`, `mask2Color` Default: `grayColor`
`w_windowId`	ID of the window in which you want to set the color for the active layer-purpose pair. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified color was set for the active layer-purpose pair.
`nil`	The command was unsuccessful.

Examples

```
pteSetActiveLppColor("mask1Color")
pteSetActiveLppColor("mask2Color" window(2))
```

pteSetActiveLppColorLock

```
pteSetActiveLppColorLock(
     { t | nil }
     [ w_windowId ]
     )
     => t / nil
```

Description

Locks or unlocks the color on the current active layer-purpose pair.

Before you can set the lock state, you must set the color to `mask1Color` or `mask2Color` by using pteSetActiveLppColor.

The `pteSetActiveLppColorLock` SKILL function does not change the lock state when using the default color. To change the lock state of the default color, use mptSetLockDefaultColors.

Arguments

`t`	Lock the color on the active layer-purpose pair.
`nil`	Unlock the color on the active layer-purpose pair.
`w_windowId`	ID of the window in which you want to lock or unlock the color specified for the active layer-purpose pair. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The color was locked or unlocked on the active layer-purpose pair.
`nil`	The command was unsuccessful.

Examples

```
pteSetActiveLppColorLock(t)
pteSetActiveLppColorLock(nil window(2))
```

pteSetAllLayerSetMember

```
pteSetAllLayerSetMember(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets all specified layer-purpose pairs as members of the layer set that is currently in use.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: Items displayed in the specified Palette panel are set as members of the current layer set. If the parent and child items in the *Objects* and *Grids* panels are in the desynchronized state, the membership state is set only for the parent items.
- `Selection`: Only the items that are selected in the specified Palette panel are set as members of the current layer set.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Membership state of both parent and child items is set. Use this mode only if the parent and child items in the *Objects* and *Grids* panels are in the desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are set as members of the current layer set.

`?panel`	Name of the Palette panel. Valid values: `Layers` and `Objects` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`w_windowId`	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All specified items were added as members to the layer set currently in use.
`nil`	The command was unsuccessful.

Examples

```
pteSetAllLayerSetMember(?panel "Layers")
pteSetAllLayerSetMember(?only t)
pteSetAllLayerSetMember(?mode "Display")
pteSetAllLayerSetMember(?window window(2))
pteSetAllLayerSetMember(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetAllLSEnable

```
pteSetAllLSEnable(
     g_showState
     [ w_windowId ]
     )
     => t / nil
```

Description

Enables or disables all the layer sets listed in the *Layer Set Manager*, except the one that is currently in use.

Arguments

g_showState	If set to `t`, all layer sets are enabled. If set to `nil`, all layer sets, except the layer set that is currently in use, are disabled.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Layer sets were enabled or disabled, as required.
`nil`	The command was unsuccessful.

Examples

```
pteSetAllLSEnable(t)
pteSetAllLSEnable(nil window(2))
```

pteSetAllSelectable

```
pteSetAllSelectable(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets as selectable the specified items in the specified Palette panel.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: All items displayed in the specified Palette panel are set as selectable. If the parent and child objects in the *Objects* panel are in a desynchronized state, only the parent items are set as selectable.
- `Selection`: Only the items selected in the specified Palette panel are set as selectable.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Both parent and child items in the specified Palette panel are set as selectable. Use this mode only if the parent and child objects in the *Objects* panel are in a desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are set as selectable.

`?panel`	Name of the Palette panel. Valid values: `Layers` and `Objects` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified items were set as selectable.
`nil`	The command was unsuccessful.

Examples

```
pteSetAllSelectable(?panel "Layers")
pteSetAllSelectable(?only t)
pteSetAllSelectable(?mode "Display")
pteSetAllSelectable(?window window(2))
pteSetAllSelectable(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetAllStipple

```
pteSetAllStipple(
     [ ?mode { Display | Selection } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets to visible the stipple for the specified layer-purpose pairs.

Arguments

`?mode`

Scope of execution of the command.

Valid values:

- `Display`: The stipple for all the layer-purpose pairs displayed in the *Layers* panel were set to visible.
- `Selection`: The stipple for the layer-purpose pairs selected in the *Layers* panel were set to visible.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.

`?only`

If set to `t`, the stipple for only the layer-purpose pairs corresponding to the specified mode were set to visible.

`?panel`

Name of the Palette panel.
Valid values: `Layers`
Default: `Layers`

`?window w_windowId`

	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The stipple for the specified layer-purpose pairs were set to visible.
`nil`	The command was unsuccessful.

Examples

```
pteSetAllStipple(?panel "Layers")
pteSetAllStipple(?only t)
pteSetAllStipple(?mode "Display")
pteSetAllStipple(?window window(2))
pteSetAllStipple(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetAllValidity

```
pteSetAllValidity(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets as valid the specified items in the specified Palette panel.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: All items displayed in the specified Palette panel are set as valid. If the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state, only the parent items are set as valid.
- `Selection`: Only the items selected in the specified Palette panel are set as valid.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Both parent and child items in the specified Palette panel are set as valid. Use this mode only if the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are set as valid.

`?panel`	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified items were set as valid.
`nil`	The command was unsuccessful.

Examples

```
pteSetAllValidity(?panel "Layers")
pteSetAllValidity(?only t)
pteSetAllValidity(?mode "Display")
pteSetAllValidity(?window window(2))
pteSetAllValidity(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetAllVisible

```
pteSetAllVisible(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets as visible the specified items in the specified Palette panel.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: All items displayed in the specified Palette panel are set as visible. If the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state, only the parent items are set as visible.
- `Selection`: Only the items selected in the specified Palette panel are set as visible.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Both parent and child items in the specified Palette panel are set as visible. Use this mode only if the parent and child objects in the *Objects* and *Grids* panels are in a desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are set as visible.

`?panel`	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified items were set as visible.
`nil`	The command was unsuccessful.

Examples

```
pteSetAllVisible(?panel "Layers")
pteSetAllVisible(?only t)
pteSetAllVisible(?mode "Display")
pteSetAllVisible(?window window(2))
pteSetAllVisible(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetBindkeys

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

Description

Opens the Bindkey Editor form to the Palette section, with the Palette bindkeys displayed in the right pane.

Arguments

None

Value Returned

`t`	The Bindkey Editor form was displayed.
`nil`	The command was unsuccessful.

Example

```
pteSetBindkeys()
```

pteSetConfig

```
pteSetConfig(
     t_configName
     t_configValue
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Assigns the specified `configValue` to the `configName` configuration parameter in the specified Palette panel.

Arguments

t_configName	Name of the Palette configuration parameter.
t_configValue	Value to be assigned to the Palette configuration parameter.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The Palette configuration parameter was set.
`nil`	The command was unsuccessful.

Examples

```
pteSetConfig("sortAscendant" "GDSNumber" "Layers")
pteSetConfig("sectionMoved" "Visibility 0" "Objects")
pteSetConfig("showTool" "Controls Buttons" "Grids")
```

pteSetFindModeOn

```
pteSetFindModeOn (
     g_FindState
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the search mode for the specified window between `Find` and `Filter`.

Arguments

g_FindState	The find state. When this argument is set to `t`, this function changes the search mode to `Find` and when set to `nil`, the search mode changes to `Filter`. Valid values: `t`, `nil`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified find state was set for all layer-purpose pairs that match the search criteria.
`nil`	The command was unsuccessful.

Examples

Changes search mode to `Find`:

```
pteSetFindModeOn(t)
```

Changes search mode to `Filter`:

```
pteSetFindModeOn(nil)
```

pteSetLayerSetColoredLpp

```
pteSetLayerSetColoredLpp(
     t_layerSetName
     t_layerPurposeName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the layer-purpose pair for the specified layer set.

Arguments

t_layerSetName	Name of the layer set.
t_layerPurposeName	Name of the layer-purpose pair.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The layer-purpose pair was set for the specified layer set.
nil	The command was unsuccessful.

Example

```
pteSetLayerSetColoredLpp("Poly" "Poly drawing")
```

pteSetLayerSetMember

```
pteSetLayerSetMember(
     t_layerPurposeName
     g_memberState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Adds or removes the specified layer-purpose pair from the layer set currently in use.

Arguments

t_layerPurposeName	Name of the layer-purpose pair to be added or removed.
g_memberState	If set to `t`, the layer-purpose pair is added as a member to the layer set currently in use. If set to `nil`, the layer-purpose pair is removed from the layer set currently in use.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer-purpose pair was added or removed from the layer set currently in use.
`nil`	The command was unsuccessful.

Examples

```
pteSetLayerSetMember("OVERLAP boundary" nil "Layers")
pteSetLayerSetMember("nwell drawing" t "Layers" window(2))
```

pteSetLppVisibleByString

```
pteSetLppVisibleByString(
     t_searchPattern
     g_visibilityState
     [ w_windowId ]
     )
     => t / nil
```

Description

Searches for the specified search string based on the advanced search options (casing, match type, and search operator) set in the *Layers* panel, and sets the visibility state for all layer-purpose pairs that match the search criteria to the specified value.

Arguments

t_searchPattern	The search string.
g_visibilityState	The visibility state. Valid values: `t`, `nil`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified visibility state was set for all layer-purpose pairs that match the search criteria.
`nil`	The command was unsuccessful.

Examples

Sets the visibility for all `Metal1` layer-purpose pairs to `nil` in the current window.

```
pteSetLppVisibleByString("Metal1" nil)
```

pteSetLSActive

```
pteSetLSActive(
     ?layerSets t_layerSets
     ?turnOnVis { t | nil }
     ?turnOnSel { t | nil }
     ?turnOffAllVis { t | nil }
     ?turnOffAllSel { t | nil }
     ?deactivateOthers { t | nil }
     [ w_windowId ]
     )
     => t / nil
```

Description

Activates one or more layer sets and can be used to force the visibility and selectability of the layer-purpose pairs contained in those layer sets. The visibility and selectability of layer-purpose pairs contained in layer sets previously set as active is affected only if the visibility or selectability of layer-purpose pairs contained in the layer set being set as active is forced.

This function is most useful when you want to activate a layer set without deactivating the layer sets previously set as active.

Arguments

`?layerSets` *`t_layerSets`*

List of layer sets to be activated.

`?turnOnVis`
If set to `t`, forces the visibility of all layer-purpose pairs in the specified layer sets. If set to `nil`, visibility of layer-purpose pairs in the layer sets being set as active remains unchanged.
Default: `nil`

`?turnOnSel`
If set to `t`, forces the selectability of all layer-purpose pairs in the specified layer sets. If set to `nil`, selectability of layer-purpose pairs in the layer sets being set as active remains unchanged.
Default: `nil`

`?turnOffAllVis`
If set to `t`, turns off the visibility of all layer-purpose pairs contained in the layer sets previously set as active. If set to `nil`, the visibility of layer-purpose pairs contained in such layer sets remains unchanged.
Default: `nil`

`?turnOffAllSel`	If set to `t`, turns off the selectability of all layer-purpose pairs contained in the layer sets previously set as active. If set to `nil`, the selectability of layer-purpose pairs contained in such layer sets remains unchanged. Default: `nil`
`?deactivateOthers`	If set to `t`, sets as inactive all layer sets previously set as active. If set to `nil`, the layer sets previously set as active do not become inactive. Default: `nil`
`w_windowId`	ID of the window containing the Palette in which you want to set a layer set as active. Default: Current window

Value Returned

`t`	The specified layer sets were activated.
`nil`	The command was unsuccessful.

Examples

The following example activates layer set *LS1*:

```
pteSetLSActive(?layerSets "LS1" ?turnOnVis nil ?turnOnSel nil ?turnOffAllVis nil
?turnOffAllSel nil ?deactivateOthers t)
```

The following example, adds layer set *LS1* to the list of active layer sets, that is, the layer sets previously set as active do not become inactive, and forces the visibility and selectability of all layer-purpose pairs contained in it. The visibility of layer-purpose pairs contained in layer sets previously set as active remains unchanged; selectability is turned off.

```
pteSetLSActive(?layerSets "LS1" ?turnOnVis t ?turnOnSel t ?turnOffAllVis nil
?turnOffAllSel t ?deactivateOthers nil)
```

The following example, adds layer set *LS1* to the list of active layer sets, that is, the layer sets previously set as active do not become inactive, and forces the visibility and selectability of all layer-purpose pairs contained in it. The visibility and selectability of layer-purpose pairs contained in layer sets previously set as active are turned off.

```
pteSetLSActive(?layerSets "LS1" ?turnOnVis t ?turnOnSel t ?turnOffAllVis t
?turnOffAllSel t ?deactivateOthers nil)
```

The following example, adds layer set *LS1* to the list of active layer sets, that is, the layer sets previously set as active do not become inactive, and keeps the visibility and selectability of

the layer-purpose pairs contained in *LS1* unchanged. The visibility and selectability of layer-purpose pairs contained in layer sets previously set as active also remain unchanged.

```
pteSetLSActive(?layerSets "LS1" ?turnOnVis nil ?turnOnSel nil ?turnOffAllVis t
?turnOffAllSel t ?deactivateOthers nil)
```

The following example, adds layer set *LS1* to the list of active layer sets, that is, the layer sets previously set as active do not become inactive, and forces the visibility of all layer-purpose pairs contained in it. The visibility of layer-purpose pairs contained in layer sets previously set as active is turned off; selectability remains unchanged.

```
pteSetLSActive(?layerSets "LS1" ?turnOnVis t ?turnOnSel nil ?turnOffAllVis t
?turnOffAllSel t ?deactivateOthers nil)
```

The following example, adds layer set *LS1* to the list of active layer sets, that is, the layer sets previously set as active do not become inactive, and forces the selectability of all layer-purpose pairs contained in it. The selectability of layer-purpose pairs contained in layer sets previously set as active is turned off; visibility remains unchanged.

```
pteSetLSActive(?layerSets "LS1" ?turnOnVis nil ?turnOnSel t ?turnOffAllVis t
?turnOffAllSel t ?deactivateOthers nil)
```

pteSetLSEnable

```
pteSetLSEnable(
     t_layerSetName
     g_showState
     [ w_windowId ]
     )
     => t / nil
```

Description

Enables or disables the specified layer set. The layer set currently in use cannot be disabled.

Arguments

t_layerSetName	Name of the layer set to be enabled or disabled.
g_showState	If set to `t`, the layer set is enabled. If set to `nil`, the layer set is disabled.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer set was enabled or disabled.
`nil`	The command was unsuccessful.

Examples

```
pteSetLSEnable("metal" t)
pteSetLSEnable("metal" nil window(2))
```

pteSetLSPosition

```
pteSetLSPosition(
     t_layerSetName
     x_position
     [ w_windowId ]
     )
     => t / nil
```

Description

Moves the specified layer set to the specified position in the *Layer Set Manager*.

Arguments

t_layerSetName	Name of the layer set that needs to be moved up or down the list.
x_position	Position to which the layer set needs to be moved, where 0 is the first item in the list, 1 the second item, and so on. Disabled layer sets, which may not be displayed in the *Layer Set Manager* at the given instance, are counted while identifying the new position for the layer set.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer set was moved to a new position.
`nil`	The command was unsuccessful.

Examples

```
pteSetLSPosition("metal" 2)
pteSetLSPosition("metal" 2 window(2))
```

pteSetLSSelectable

```
pteSetLSSelectable(
     t_layerSetName
     g_selectabilityState
     [ w_windowId ]
     )
     => t / nil
```

Description

Turns on or off the selectability state of all the layer-purpose pairs contained in the specified layer set.

Arguments

t_layerSetName	Name of the layer set for which selectability is to be turned on or off.
g_selectabilityState	If set to `t`, selectability is turned on. If set to `nil`, selectability is turned off.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The selectability of all the layer-purpose pairs contained in the specified layer set was turned on or off.
`nil`	The command was unsuccessful.

Examples

```
pteSetLSSelectable("metal" t)
pteSetLSSelectable("metal" nil window(2))
```

pteSetLSVisible

```
pteSetLSVisible(
     t_layerSetName
     g_visibilityState
     [ w_windowId ]
     )
     => t / nil
```

Description

Turns on or off the visibility state of all the layer-purpose pairs contained in the specified layer set.

Arguments

t_layerSetName	Name of the layer set for which visibility is to be turned on or off.
g_visibilityState	If set to `t`, visibility is turned on. If set to `nil`, visibility is turned off.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The visibility of all the layer-purpose pairs contained in the specified layer set was turned on or off.
`nil`	The command was unsuccessful.

Examples

```
pteSetLSVisible("metal" t)
pteSetLSVisible("metal" nil window(2))
```

pteSetMPTSupportMode

```
pteSetMPTSupportMode(
     w_windowId
     { t | nil }
     )
     => t / nil
```

Description

Enables or disables MPT support in the window with the specified ID.

Arguments

w_windowId	ID of any open window.
t	Enable MPT support in the window with the specified ID.
nil	Disable MPT support in the window with the specified ID.

Value Returned

t	MPT support was enabled or disabled in the window with the specified ID.
nil	The command was unsuccessful.

Examples

```
pteSetMPTSupportMode(hiGetCurrentWindow() t)
pteSetMPTSupportMode(window(3) nil)
```

pteSetNoneLayerSetMember

```
pteSetNoneLayerSetMember(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Removes from the active layer set all items that meet the specified criteria.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: Items displayed in the specified Palette panel are removed from the active layer set. If the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state, only the parent objects or grids are removed from the active layer set.
- `Selection`: The items that are selected in the specified Palette panel are removed from the active layer set.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Both parent and child items are removed from the active layer set. Use this mode only if the parent and child items in the *Objects* and *Grids* panels are in the desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are removed from the layer set.

`?panel`	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified layer-purpose pairs were removed from the active layer set.
`nil`	The command was unsuccessful.

Examples

```
pteSetNoneLayerSetMember(?panel "Layers")
pteSetNoneLayerSetMember(?only t)
pteSetNoneLayerSetMember(?mode "Display")
pteSetNoneLayerSetMember(?window window(2))
pteSetNoneLayerSetMember(?mode "Display" ?only t ?panel "Layers" ?window
window(2))
```

pteSetNoneSelectable

```
pteSetNoneSelectable(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Turns off the selectability of the specified items in the specified Palette panel.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: The selectability of all the items displayed in the specified Palette panel is turned off. If the parent and child objects in the *Objects* panel are in a desynchronized state, the selectability of only the parent items is turned off.
- `Selection`: The selectability of only the items selected in the specified Palette panel is turned off.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: The selectability of both parent and child items in the specified Palette panel is turned off. Use this mode only if the parent and child objects in the *Objects* panel are in a desynchronized state.

`?only` — If set to `t`, the selectability of only the items corresponding to the specified mode is turned off.

`?panel`	Name of the Palette panel. Valid values: `Layers` and `Objects` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The selectability of the specified items was turned off.
`nil`	The command was unsuccessful.

Examples

```
pteSetNoneSelectable(?panel "Layers")
pteSetNoneSelectable(?only t)
pteSetNoneSelectable(?mode "Display")
pteSetNoneSelectable(?window window(2))
pteSetNoneSelectable(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetNoneStipple

```
pteSetNoneStipple(
     [ ?mode { Display | Selection } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Hides the stipple for the specified layer-purpose pairs.

Arguments

`?mode`	Scope of execution of the command. Valid values: ■ `Display`: The stipple for all the layer-purpose pairs displayed in the *Layers* Palette panel are hidden. ■ `Selection`: The stipple for the layer-purpose pairs selected in the *Layers* panel are hidden. **Note:** The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
`?only`	If set to `t`, the stipple for only the layer-purpose pairs corresponding to the specified mode are hidden.
`?panel`	Name of the Palette panel. Valid values: `Layers` Default: `Layers`
`?window w_windowId`	

ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	The stipple for the specified layer-purpose pairs were hidden.
`nil`	The command was unsuccessful.

Examples

```
pteSetNoneStipple(?panel "Layers")
pteSetNoneStipple(?only t)
pteSetNoneStipple(?mode "Display")
pteSetNoneStipple(?window window(2))
pteSetNoneStipple(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetNoneValidity

```
pteSetNoneValidity(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets as invalid the specified items in the specified Palette panel.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: All items displayed in the specified Palette panel are set as invalid. If the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state, only the parent items are set as invalid.
- `Selection`: Only the items selected in the specified Palette panel are set as invalid.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Both parent and child items in the specified Palette panel are set as invalid. Use this mode only if the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are set as invalid.

`?panel`	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified items were set as invalid.
`nil`	The command was unsuccessful.

Examples

```
pteSetNoneValidity(?panel "Layers")
pteSetNoneValidity(?only t)
pteSetNoneValidity(?mode "Display")
pteSetNoneValidity(?window window(2))
pteSetNoneValidity(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetNoneVisible

```
pteSetNoneVisible(
     [ ?mode { Display | Selection | Propagate } ]
     [ ?only { t | nil } ]
     [ ?panel { Layers | Objects | Grids } ]
     [ ?window w_windowId ]
     )
     => t / nil
```

Description

Sets as invisible the specified items in the specified Palette panel.

Arguments

`?mode` — Scope of execution of the command.

Valid values:

- `Display`: All items displayed in the specified Palette panel are set as invisible. If the parent and child items in the *Objects* and *Grids* panels are in a desynchronized state, only the parent items are set as invisible.
- `Selection`: Only the items selected in the specified Palette panel are set as invisible.

  Note: The Selection option is applicable when multi-select mode is selected in Palette Options form or it is enabled using the environment variable, `pteMultiSelectionMode`. This lets you select multiple LPPs in the Layers panel using the `CTRL` + `LMB` or `Shift` + `LMB` after an initial `LMB` click on an LPP. The Selection mode in the active command applies to the selected LPPs *Layers* panel.
- `Propagate`: Both parent and child items in the specified Palette panel are set as invisible. Use this mode only if the parent and child objects in the *Objects* and *Grids* panels are in a desynchronized state.

`?only` — If set to `t`, only the items corresponding to the specified mode are set as invisible.

`?panel`	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
`?window` *`w_windowId`*	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified items were set as invisible.
`nil`	The command was unsuccessful.

Examples

```
pteSetNoneVisible(?panel "Layers")
pteSetNoneVisible(?only t)
pteSetNoneVisible(?mode "Display")
pteSetNoneVisible(?window window(2))
pteSetNoneVisible(?mode "Display" ?only t ?panel "Layers" ?window window(2))
```

pteSetOnlySelectable

```
pteSetOnlySelectable(
     { t_layerPurposeName | t_paletteElementPath }
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Turns on the selectability of the specified item in the specified panel. All other items in the specified panel are made non-selectable.

Arguments

t_layerPurposeName	Name of the item to be made selectable.
t_paletteElementPath	Path to the item to be made selectable.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Selectability of the specified item was turned on.
`nil`	The command was unsuccessful.

Examples

```
pteSetOnlySelectable("Nwell drawing" "Layers")
pteSetOnlySelectable("Oxide drawing" "Layers" window(2))
```

pteSetOnlySelectableWithDepth

```
pteSetOnlySelectableWithDepth(
     { t_layerPurposeName | t_paletteElementPath }
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Turns on the selectability of the specified layer-purpose pair and of the *n* layer-purpose pairs listed both above and below the specified layer-purpose pair in the *Layers* panel. The value *n* is obtained from the *Depth for Selectability* field in the Palette Options form. Selectability of all other layer-purpose pairs is turned off.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_paletteElementPath

Path to the layer-purpose pair.

t_panelName	Name of the Palette panel. Valid values: `Layers`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Selectability of the specified layer-purpose pairs was turned on.
`nil`	The command was unsuccessful.

Examples

```
pteSetOnlySelectableWithDepth("Nwell drawing" "Layers")
```

```
pteSetOnlySelectableWithDepth("Oxide drawing" "Layers" window(2))
```

pteSetOnlyVisible

```
pteSetOnlyVisible(
     { t_layerPurposeName | t_paletteElementPath }
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Turns on the visibility of the specified item in the specified panel. All other items are made invisible.

Arguments

t_layerPurposeName	Name of the item to be made visible.
t_paletteElementPath	Path to the item to be made visible.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Visibility of the specified item was turned on.
`nil`	The command was unsuccessful.

Examples

```
pteSetOnlyVisible("Nwell drawing" "Layers")
pteSetOnlyVisible("Oxide drawing" "Layers" window(2))
```

pteSetOnlyVisibleWithDepth

```
pteSetOnlyVisibleWithDepth(
     { t_layerPurposeName | t_paletteElementPath }
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Turns on the visibility of the specified layer-purpose pair and of the *n* layer-purpose pairs listed both above and below the specified layer-purpose pair in the *Layers* panel. The value *n* is obtained from the *Depth for Visibility* field in the Palette Options form. Visibility of all other layer-purpose pairs is turned off.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_paletteElementPath

Path to the layer-purpose pair.

t_panelName	Name of the Palette panel. Valid values: `Layers`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	Visibility of the specified layer-purpose pairs was turned on.
`nil`	The command was unsuccessful.

Examples

```
pteSetOnlyVisibleWithDepth("Nwell drawing" "Layers")

pteSetOnlyVisibleWithDepth("Oxide drawing" "Layers" window(2))
```

pteSetOption

```
pteSetOption(
     t_optionName
     g_optionState
     [ w_windowId ]
     )
     => t / nil
```

Description

Assigns a value to the specified Palette option.

Arguments

t_optionName	Name of the Palette option for which a value is to be assigned. Valid value: `contextByLS`.
g_optionState	Value to be assigned to the Palette option.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified value was ed to the Palette option.
`nil`	The command was unsuccessful.

Examples

Assigns the specified value to the Palette option.

```
pteSetOption("contextByLS" t)
pteSetOption("contextByLS" t window(2))
```

pteSetOptionString

```
pteSetOptionString(
     t_envName
     t_envValue
     [ w_windowId ]
     )
     => t / nil
```

Description

Changes the state of a Palette environment variable during a Virtuoso session.

Arguments

t_envName	Name of the Palette environment variable. Valid values: `pteFindModeOn`, `pteMultiSelectionMode`, `pteAutoRedrawMode`, `pteWindowSynchronizationMode`, `pteAllLayersScope`, `pteCheckableColumnClickMode`, `pteLayerSetManagerLeftClickMode`, `pteLayerSetActivationMode`, `pteUsedLayersEipScope`, `pteInverseColorScheme`, `pteMovableTools`, `pteApplyParentState`, `pteRoutingOnlyScope`, `pteUsedOnlyScope`, `pteValidOnlyScope`, `pteLayersVisibilityDepth`, `pteLayersSelectabilityDepth`, `pteLinkColorVisibilityToShapeVisibility`, `pteColorMixedSort`, and `pteCompactMPT` A change to the following environment variables is propagated to open windows in real time: `pteMultiSelectionMode`, `pteCheckableColumnClickMode`, `pteLayerSetManagerLeftclickMode`, `pteAllLayersScope`, `pteUsedLayersEipScope`, `pteInverseColorScheme`, `pteMovableTools`, `pteApplyParentState`, `pteLayersVisibilityDepth`, `pteLayersSelectabilityDepth`, `pteLinkColorVisibilityToShapeVisibility`, `pteColorMixedSort`, and `pteCompactMPT`. Other environment variables are evaluated only when a new window is opened.
t_envValue	Value of the Palette environment variable to be set.

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window

Value Returned

t	The specified value was assigned to the Palette environment variable.
nil	The command was unsuccessful.

Examples

```
pteSetOptionString("pteFindModeOn" "t")
pteSetOptionString("pteFindModeOn" "nil")

pteSetOptionString("pteMultiSelectionMode" "t")
pteSetOptionString("pteMultiSelectionMode" "nil")

pteSetOptionString("pteAutoRedrawMode" "noWindow")
pteSetOptionString("pteAutoRedrawMode" "allWindows")
pteSetOptionString("pteAutoRedrawMode" "currentWindow")

pteSetOptionString("pteWindowSynchronizationMode" "independent")
pteSetOptionString("pteWindowSynchronizationMode" "synchronized")

pteSetOptionString("pteAllLayersScope" "techfileLayers")
pteSetOptionString("pteAllLayersScope" "lswLayers")

pteSetOptionString("pteCheckableColumnClickMode" "sort")
pteSetOptionString("pteCheckableColumnClickMode" "turnOnOff")

pteSetOptionString("pteLayerSetManagerLeftClickMode"
"forceVisibilityAndSelectabilityOnly")
pteSetOptionString("pteLayerSetManagerLeftClickMode" "activateLayerSet")

pteSetOptionString("pteLayerSetActivationMode" "filterAndApply")
pteSetOptionString("pteLayerSetActivationMode" "filterOnly")

pteSetOptionString("pteUsedLayersEipScope" "topToBottom")
pteSetOptionString("pteUsedLayersEipScope" "currentToBottom")
```

```
pteSetOptionString("pteInverseColorScheme" "t")
pteSetOptionString("pteInverseColorScheme" "nil")

pteSetOptionString("pteMovableTools" "t")
pteSetOptionString("pteMovableTools" "nil")

pteSetOptionString("pteApplyParentState" "t")
pteSetOptionString("pteApplyParentState" "nil")

pteSetOptionString("pteRoutingOnlyScope" "t")
pteSetOptionString("pteRoutingOnlyScope" "nil")

pteSetOptionString("pteUsedOnlyScope" "t")
pteSetOptionString("pteUsedOnlyScope" "nil")

pteSetOptionString("pteValidOnlyScope" "t")
pteSetOptionString("pteValidOnlyScope" "nil")

pteSetOptionString("pteLayersVisibilityDepth" "5")

pteSetOptionString("pteLayersSelectabilityDepth" "4")

pteSetOptionString("pteLinkColorVisibilityToShapeVisibility" "t")
pteSetOptionString("pteLinkColorVisibilityToShapeVisibility" "nil")

pteSetOptionString("pteColorMixedSort" "t")
pteSetOptionString("pteColorMixedSort" "nil")

pteSetOptionString("pteCompactMPT" "t")
pteSetOptionString("pteCompactMPT" "nil")
```

pteSetRoutingDirection

```
pteSetRoutingDirection(
     t_layerPurposeName
     t_routingDirectionName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the routing direction for the specified layer-purpose pair.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_routingDirectionName

Routing direction to be set for the layer-purpose pair.
Valid values: `Left Diagonal`, `Right Diagonal`, `Horizontal`, `Vertical`, and `None`

w_windowId

ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	The specified routing direction was set.
`nil`	The command was unsuccessful.

Examples

```
pteSetRoutingDirection("Metal1 drawing" "Left Diagonal")
pteSetRoutingDirection("Metal1 drawing" "None" window(2))
```

pteSetSearchMatchCase

```
pteSetSearchMatchCase(
     t_matchCaseType
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Specifies whether search is case-sensitive or case-insensitive.

Arguments

t_matchCaseType	If set to `sensitive`, the casing of the search string is considered. If set to `insensitive`, the casing of the search string is ignored. Default: `insensitive`
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified option was set.
`nil`	The command was unsuccessful.

Examples

Sets the search function to perform case-sensitive search in the *Layers* panel:

```
pteSetSearchMatchCase("sensitive" "Layers")
```

Sets the search function to perform case-insensitive search in the *Grids* panel in the window with ID 2:

```
pteSetSearchMatchCase("insensitive" "Grids" window(2))
```

pteSetSearchMatchType

```
pteSetSearchMatchType(
     t_searchType
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Limits searches to items that match the search string, items with a specific prefix or suffix, or items that contain the search string as a substring.

Arguments

t_searchType	Scope of the search string. Valid values: `substring`, `prefix`, `exactly`, and `suffix` Default: `substring`
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The scope of the search string was set.
`nil`	The command was unsuccessful.

Examples

Sets the search function to retrieve the layer-purpose pairs that contain the specified search string:

```
pteSetSearchMatchType("substring" "Layers")
```

Sets the search function to retrieve grid objects that match the specified search string:

```
pteSetSearchMatchType("exactly" "Grids" window(2))
```

pteSetSearchOperator

```
pteSetSearchOperator
     t_searchOperator
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the default operator for the search criteria. Valid values are `and`, `or`, `phrase`, and `not`. Panel name is one of `Layers`, `Objects`, or `Grids`.

Arguments

t_searchOperator	Operator for the search operation. Valid values: `and` (*All Of The Words*), `or` (*Any Of The Words*), `phrase` (*The Exact Phrase*), `not` (*None Of The Words*) Default: `and`
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified search operator was set.
`nil`	The command was unsuccessful.

Examples

Sets the search function to retrieve layer-purpose pairs that match any of the words in the search criteria:

```
pteSetSearchOperator("or" "Layers")
```

Sets the search function to retrieve grid items that contain none of the words specified in the search criteria:

```
pteSetSearchOperator("not" "Grids" window(2))
```

pteSetSearchText

```
pteSetSearchText(
     t_searchPattern
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Searches for the specified search string based on the casing, match type, and search operator set for the panel in which the search is to be performed.

Arguments

t_searchPattern	Search string for which search is performed.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The items matching the specified search string were listed in the specified panel.
`nil`	The command was unsuccessful.

Examples

Displays only those layer-purpose pairs that start with the letter `M`.

```
pteSetSearchMatchCase("sensitive" "Layers")
pteSetSearchMatchType("prefix" "Layers")
pteSetSearchOperator("or" "Layers")
pteSetSearchText("M" "Layers")
```

pteSetSelectable

```
pteSetSelectable(
     { t_layerPurposeName | t_paletteElementPath }
     g_selectabilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the selectability state for the specified item.

Arguments

t_layerPurposeName

Name of the item to be made selectable.

t_paletteElementPath

Path to the item to be made selectable.

g_selectabilityState

Selectability state to be set for the specified item. If set to `t`, selectability is turned on. If set to `nil`, selectability is turned off.

t_panelName

Name of the Palette panel.
Valid values: `Layers`, `Objects`, and `Grids`

w_windowId

ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`

The selectability state was set for the specified item.

`nil`

The command was unsuccessful.

Examples

```
pteSetSelectable("Metal1 drawing" nil "Layers")
pteSetSelectable("Metal1;Metal1 drawing" nil "Layers") (in ViewBy mode)
pteSetSelectable("Shapes;Circle/Ellipse" nil "Objects" window(2))
```

Related Topics

pteIsSelectable

pteSetSelectableWithDepth

```
pteSetSelectableWithDepth(
     { t_layerPurposeName | t_paletteElementPath }
     g_selectabilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the selectability for the specified layer-purpose pair and for the `n` layer-purpose pairs listed both above and below the specified layer-purpose pair in the *Layers* panel. The value `n` is obtained from the *Depth for Selectability* field in the Palette Options form.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_paletteElementPath

Path to the layer-purpose pair.

g_selectabilityState

Selectability state to be set for the specified layer-purpose pairs. If set to `t`, selectability is turned on. If set to `nil`, selectability is turned off.

t_panelName

Name of the Palette panel.
Valid values: `Layers`

w_windowId

ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`

The selectability state was set for the specified layer-purpose pairs.

`nil`

The command was unsuccessful.

Examples

```
pteSetSelectableWithDepth("Metal1 drawing" nil "Layers")
pteSetSelectableWithDepth("Metal1;Metal1 drawing" nil "Layers") (in ViewBy mode)
```

pteSetShowLockedColors

```
pteSetShowLockedColors(
     { t | nil }
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays or hides the colors locked on shapes.

Arguments

`t`	Displays the colors locked on shapes. This is the default value.
`nil`	Hides the colors locked on shapes.
`w_windowId`	ID of the window in which you want to display or hide locked colors. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The colors locked on shapes were displayed or hidden.
`nil`	The command was unsuccessful.

Examples

```
pteSetShowLockedColors(t)
pteSetShowLockedColors(nil window(2))
```

pteSetShowUnlockedColors

```
pteSetShowUnlockedColors(
     { t | nil }
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays or hides the unlocked colors on shapes.

Arguments

`t`	Displays the unlocked colors on shapes. This is the default value.
`nil`	Hides the unlocked colors on shapes.
`w_windowId`	ID of the window in which you want to display or hide unlocked colors. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The unlocked colors on shapes were displayed or hidden.
`nil`	The command was unsuccessful.

Examples

```
pteSetShowUnlockedColors(t)
pteSetShowUnlockedColors(nil window(2))
```

pteSetStipple

```
pteSetStipple(
     { t_layerPurposeName | t_paletteElementPath }
     g_stippleVisibilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the stipple visibility state for the specified layer-purpose pair.

Arguments

t_layerPurposeName

Name of the layer-purpose pair for which the stipple visibility state is to be set.

t_paletteElementPath

Path to the layer-purpose pair for which the stipple visibility state is to be set.

g_stippleVisibilityState

Stipple visibility state to be set for the specified layer-purpose pair. If set to `t`, stipple visibility is turned on. If set to `nil`, stipple visibility is turned off.

t_panelName
Name of the Palette panel.
Valid values: `Layers`

w_windowId
ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`
The stipple visibility state was set for the specified layer-purpose pair.

`nil`
The command was unsuccessful.

Examples

```
pteSetStipple("Metal1 drawing" nil "Layers")
pteSetStipple("Metal1;Metal1 drawing" nil "Layers") (in ViewBy mode)
pteSetStipple("Metal1 drawing" t "Layers" window(2))
```

pteSetValidity

```
pteSetValidity(
     { t_layerPurposeName | t_paletteElementPath }
     g_validityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the validity state for the specified item.

Arguments

t_layerPurposeName

Name of the item to be made valid.

t_paletteElementPath

Path to the item to be made valid.

g_validityState	Validity state to be set for the specified layer-purpose pair or Palette element. If set to `t`, validity is turned on. If set to `nil`, validity is turned off.
t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The validity state was set for the specified item.
`nil`	The command was unsuccessful.

Examples

```
pteSetValidity("Metal8 drawing" nil "Layers")
```

```
pteSetValidity("Metal8 drawing" nil "Layers" window(2))
```

pteSetVisible

```
pteSetVisible(
     { t_layerPurposeName | t_paletteElementPath }
     g_visibilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the visibility state for the specified item.

Arguments

t_layerPurposeName

Name of the item to be made visible.

t_paletteElementPath

Path to the item to be made visible.

g_visibilityState

Visibility state to be set for the specified item. If set to `t`, visibility is turned on. If set to `nil`, visibility is turned off.

t_panelName — Name of the Palette panel.
Valid values: `Layers`, `Objects`, and `Grids`

w_windowId — ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	The visibility state was set for the specified item.
`nil`	The command was unsuccessful.

Examples

```
pteSetVisible("Metal1 drawing" nil "Layers")
```

```
pteSetVisible("Metal1;Metal1 drawing" nil "Layers") (in ViewBy mode)
pteSetVisible("Metal1" nil "Layers") (in ViewBy mode)
pteSetVisible("Shapes;Circle/Ellipse" nil "Objects" window(2))
```

Related Topics

pteIsVisible

pteSetVisibleWithDepth

```
pteSetVisibleWithDepth(
     { t_layerPurposeName | t_paletteElementPath }
     g_visibilityState
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Sets the visibility for the specified layer-purpose pair and for the *n* layer-purpose pairs listed both above and below the specified layer-purpose pair in the *Layers* panel. The value *n* is obtained from the *Depth for Visibility* field in the Palette Options form.

Arguments

t_layerPurposeName

Name of the layer-purpose pair.

t_paletteElementPath

Path to the layer-purpose pair.

g_visibilityState

Visibility state to be set for the specified layer-purpose pairs. If set to `t`, visibility is turned on. If set to `nil`, visibility is turned off.

t_panelName

Name of the Palette panel.
Valid values: `Layers`

w_windowId

ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	The visibility state was set for the specified layer-purpose pairs.
`nil`	The command was unsuccessful.

Examples

```
pteSetVisibleWithDepth("Metal1 drawing" nil "Layers")
pteSetVisibleWithDepth("Metal1;Metal1 drawing" nil "Layers") (in ViewBy mode)
```

pteSetWindowSynchro

```
pteSetWindowSynchro(
     g_synchronizedState
     [ w_windowId ]
     )
     => t / nil
```

Description

Synchronizes or desynchronizes the Palette.

Arguments

g_synchronizedState

	If set to `t`, the Palette is synchronized. If set to `nil`, the Palette is desynchronized.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The Palette was synchronized or desynchronized, as required.
`nil`	The command was unsuccessful.

Examples

```
pteSetWindowSynchro(t)
pteSetWindowSynchro(t window(2))
```

pteShowAllTools

```
pteShowAllTools(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays all the toolbars in the specified Palette panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	All toolbars in the specified Palette panel were displayed.
`nil`	The command was unsuccessful.

Examples

```
pteShowAllTools("Layers")
pteShowAllTools("Layers" window(2))
```

pteShowMPTLPP

```
pteShowMPTLPP(
     { t | nil }
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays in the *Layers* panel only those layer-purpose pairs that have color information.

Arguments

`t`	Displays only those layer-purpose pairs that have color information.
`nil`	Displays all layer-purpose pairs.
w_windowId	ID of the window in which you want to display the layer-purpose pairs that have color information. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The required layer-purpose pairs were displayed.
`nil`	The command was unsuccessful.

Examples

```
pteShowMPTLPP(t)
pteShowMPTLPP(nil window(2))
```

pteShowRoutingLPP

```
pteShowRoutingLPP(
     g_filterState
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays only the routing layer-purpose pairs in the *Layers* panel.

Arguments

g_filterState	If set to `t`, only the routing layer-purpose pairs are displayed. If set to `nil`, all layer-purpose pairs are displayed.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The required layer-purpose pairs were displayed.
`nil`	The command was unsuccessful.

Examples

```
pteShowRoutingLPP(t)
pteShowRoutingLPP(t window(2))
```

pteShowUsedLPP

```
pteShowUsedLPP(
     g_filterState
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays only the currently used layer-purpose pairs in the *Layers* panel.

Arguments

g_filterState	If set to `t`, only the layer-purpose pairs that are currently in use are displayed. If set to `nil`, all layer-purpose pairs are displayed.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The required layer-purpose pairs were displayed.
`nil`	The command was unsuccessful.

Examples

```
pteShowUsedLPP(t)
pteShowUsedLPP(t window(2))
```

pteShowValidLPP

```
pteShowValidLPP(
     g_filterState
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays only valid layer-purpose pairs in the *Layers* panel.

Arguments

g_filterState	If set to `t`, only valid layer-purpose pairs are displayed. If set to `nil`, all layer-purpose pairs are displayed.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The required layer-purpose pairs were displayed.
`nil`	The command was unsuccessful.

Examples

```
pteShowValidLPP(t)
pteShowValidLPP(t window(2))
```

pteToggleAllLayerSetMember

```
pteToggleAllLayerSetMember(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the layer set membership status of all the items in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer set membership status of all the items in the specified panel was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleAllLayerSetMember("Grids")
pteToggleAllLayerSetMember("Objects" window(2))
```

pteToggleAllSelectable

```
pteToggleAllSelectable(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the selectability status of all the items listed in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers` and `Objects` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The selectability status of all the items in the specified panel was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleAllSelectable("Objects")
pteToggleAllSelectable("Objects" window(2))
```

pteToggleAllStipple

```
pteToggleAllStipple(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the stipple visibility status of all the layer-purpose pairs in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers` Default: `Layers`
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The stipple visibility status of all layer-purpose pairs was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleAllStipple("Layers")
pteToggleAllStipple("Layers" window(2))
```

pteToggleAllTools

```
pteToggleAllTools(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the display status of all toolbars in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The display status of all the toolbars was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleAllTools("Layers")
pteToggleAllTools("Objects" window(2))
```

pteToggleAllValidity

```
pteToggleAllValidity(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the validity status of all the items in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The validity status of all the items in the specified panel was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleAllValidity("Layers")
pteToggleAllValidity("Layers" window(2))
```

pteToggleAllVisible

```
pteToggleAllVisible(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the visibility status of all the items listed in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The visibility status of all the items in the specified panel was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleAllVisible("Grids")
pteToggleAllVisible("Objects" window(2))
```

pteToggleLayerSetEdition

```
pteToggleLayerSetEdition(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays or hides the membership status of layer-purpose pairs.

Arguments

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The membership status of layer-purpose pairs was displayed or hidden, as required.
nil	The command was unsuccessful.

Examples

```
pteToggleLayerSetEdition()
pteToggleLayerSetEdition(window(2))
```

pteToggleLayerSetValidityEdition

```
pteToggleLayerSetValidityEdition(
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays or hides the validity and membership status of layer-purpose pairs.

Arguments

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The validity and membership status of layer-purpose pairs was displayed or hidden, as required.
nil	The command was unsuccessful.

Examples

```
pteToggleLayerSetValidityEdition()
pteToggleLayerSetValidityEdition(window(2))
```

pteToggleLSActive

```
pteToggleLSActive(
     ?layerSets t_layerSetName
     [ ?toggleVis { t | nil } ]
     [ ?toggleSel { t | nil } ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Adds or removes the specified layer set from the list of active layer sets. Additionally, toggles the visibility and selectability status of the various layer-purpose pairs contained in the layer set.

Arguments

`?layerSets` *t_layerSetName*

Layer set name.

`?toggleVis`
If set to `t`, the visibility status of all layer-purpose pairs contained in the layer set is toggled.
If set to `nil`, the visibility of all layer-purpose pairs contained in the layer set remains unchanged.

`?toggleSel`
If set to `t`, the selectability status of all layer-purpose pairs contained in the layer set is toggled.
If set to `nil`, the selectability of all layer-purpose pairs contained in the layer set remains unchanged.

w_windowId
ID of the window containing the Palette that you want to update.
Valid values: ID of any open window
Default: Current window

Value Returned

`t`	The specified layer set was added or removed from the list of active layers and the visibility and selectability status was toggled as required.
`nil`	The command was unsuccessful.

Examples

```
pteToggleLSActive(?layerSet "All Layers" ?toggleVis t ?toggleSel nil)
pteToggleLSActive(?layerSet pteGetLSAtPosition(1) ?toggleVis t ?toggleSel t
window(2))
```

pteTogglePanels

```
pteTogglePanels(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Displays the contents of the specified panel and hides the other two panels.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified panel was displayed and the other two panels were hidden.
`nil`	The command was unsuccessful.

Examples

```
pteTogglePanels("Grids")
pteTogglePanels("Objects" window(2))
```

pteTogglePropagateAllSelectable

```
pteTogglePropagateAllSelectable(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the selectability status of all parent and child items in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers` and `Objects` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The selectability status of parent and child items in the specified panel was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteTogglePropagateAllSelectable("Objects")
pteTogglePropagateAllSelectable("Objects" window(2))
```

pteTogglePropagateAllVisible

```
pteTogglePropagateAllVisible(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the visibility status of all parent and child items in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The visibility status of parent and child items in the specified panel was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteTogglePropagateAllVisibile("Grids")
pteTogglePropagateAllVisibile("Objects" window(2))
```

pteToggleShowMPTLPP

```
pteToggleShowMPTLPP (
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the display of layer-purpose pairs in the *Layers* panel between all layer-purpose pairs and layer-purpose pairs with color information.

Arguments

w_windowId	ID of the window in which you want to toggle the display of layer-purpose pairs. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer-purpose pair display was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleShowMPTLPP()
pteToggleShowMPTLPP(window(2))
```

pteToggleShowRoutingLPP

```
pteToggleShowRoutingLPP(
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the display of layer-purpose pairs in the *Layers* panel between routing layer-purpose pairs and all layer-purpose pairs.

Arguments

w_windowId	ID of the window in which you want to toggle the display of layer-purpose pairs. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer-purpose pair display was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleShowRoutingLPP()
pteToggleShowRoutingLPP(window(2))
```

pteToggleShowUsedLPP

```
pteToggleShowUsedLPP(
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the display of layer-purpose pairs in the *Layers* panel between currently used layer-purpose pairs and all layer-purpose pairs.

Arguments

w_windowId	ID of the window in which you want to toggle the display of layer-purpose pairs. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer-purpose pair display was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleShowUsedLPP()
pteToggleShowUsedLPP(window(2))
```

pteToggleShowValidLPP

```
pteToggleShowValidLPP(
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the display of layer-purpose pairs in the *Layers* panel between valid layer-purpose pairs and all layer-purpose pairs.

Arguments

w_windowId	ID of the window in which you want to toggle the display of layer-purpose pairs. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The layer-purpose pair display was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleShowValidLPP()
pteToggleShowValidLPP(window(2))
```

pteToggleToolbars

```
pteToggleToolbars(
     t_panelName
     [ w_windowId ]
     )
     => t / nil
```

Description

Toggles the display status of all toolbars in the specified panel.

Arguments

t_panelName	Name of the Palette panel. Valid values: `Layers`, `Objects`, and `Grids` Default: The panel that is currently active. If none of the panels is active, the command is run on the `Layers` panel.
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The display status of all the toolbars was toggled.
`nil`	The command was unsuccessful.

Examples

```
pteToggleToolbars("Grids")
pteToggleToolbars("Objects" window(2))
```

pteToggleWindowSynchro

```
pteToggleWindowSynchro(
     [ w_windowId ]
     )
     => t / nil
```

Description

Synchronizes or desynchronizes the Palette.

Arguments

w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

t	The synchronization status of the Palette was toggled.
nil	The command was unsuccessful.

Examples

```
pteToggleWindowSynchro()
pteToggleWindowSynchro(window(2))
```

pteUndockWindow

```
pteUndockWindow(
     t_assistantName
     [ w_windowId ]
     )
     => t / nil
```

Description

Undocks the specified Palette panel.

Arguments

t_assistantName	Name of the panel to be undocked. Valid values: `Layers`, `Objects`, or `Grids` in `SingleAssistant` mode
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified panel was undocked.
`nil`	The command was unsuccessful.

Examples

```
pteUndockWindow("Layers")
pteUndockWindow("Layers" window(2))
```

pteUnmapWindow

```
pteUnmapWindow(
     t_assistantName
     [ w_windowId ]
     )
     => t / nil
```

Description

Unmaps (hides, if displayed) the specified Palette panel.

Arguments

t_assistantName	Name of the panel to be hidden from view. Valid values: `Layers`, `Objects`, or `Grids` in `SingleAssistant` mode
w_windowId	ID of the window containing the Palette that you want to update. Valid values: ID of any open window Default: Current window

Value Returned

`t`	The specified Palette panel was hidden from view.
`nil`	The command was unsuccessful.

Examples

```
pteUnmapWindow("Layers")
pteUnmapWindow("Layers" window(2))
```

pteUnRegisterUserLSManagerTrigger

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

Description

Unregisters the SKILL procedure that was registered using `pteRegisterUserLSManagerTrigger`.

Arguments

None

Value Returned

`t`	The trigger was successfully unregistered.
`nil`	The command was unsuccessful.

Example

```
pteUnRegisterUserLSManagerTrigger()
```

pteUnRegisterUserSelectionTrigger

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

Description

Unregisters the SKILL procedure that was registered using `pteRegisterUserSelectionTrigger`.

Arguments

None

Value Returned

`t`	The trigger was successfully unregistered.
`nil`	The command was unsuccessful.

Example

```
pteUnRegisterUserSelectionTrigger()
```

pteUnsetLSActive

```
pteUnsetLSActive(
     ?layerSets t_layerSets
     ?turnOffAllVis { t | nil }
     ?turnOffAllSel { t | nil }
     [ w_windowId ]
     )
     => t / nil
```

Description

Deactivates one or more layer sets and can be used to force the visibility and selectability of the layer-purpose pairs contained in those layer sets.

Arguments

`?layerSets` *t_layerSets*

List of layer sets to be deactivated.

`?turnOffAllVis`
If set to `t`, turns off the visibility of all layer-purpose pairs contained in the layer sets previously set as active. If set to `nil`, the visibility of layer-purpose pairs contained in such layer sets remains unchanged.
Default: `nil`

`?turnOffAllSel`
If set to `t`, turns off the selectability of all layer-purpose pairs contained in the layer sets previously set as active. If set to `nil`, the selectability of layer-purpose pairs contained in such layer sets remains unchanged.
Default: `nil`

w_windowId
ID of the window containing the Palette in which you want to set a layer set as active.
Default: Current window

Value Returned

`t` The specified layer sets were deactivated.

`nil` The command was unsuccessful.

Example

```
pteUnsetLSActive(?layerSets "layerSetName2" ?turnOffVis t ?turnOffSel t)
```

FinFET and Width Spacing Pattern Functions

To manage snap patterns, you can use the SKILL functions associated with FinFET and Width Spacing Pattern in Virtuoso Layout Suite. For example, leGetGlobalGridsVisible returns the visibility setting of global snap pattern grids and leGetWidthSpacingGridsVisible returns the visibility setting of width spacing snap pattern grids.

Related Topics

leCycleSnapPatternDisplay

leGetGlobalGridsVisible

leGetLocalGridsVisible

leGetSnapPatternVisible

leGetSnapToSPTransform

leCycleSnapPatternDisplay

```
leCycleSnapPatternDisplay(
     [ w_windowId ]
     )
     => t_snapPatternDisplayMode
```

Description

Sets the snap pattern display to the next value in the *Snap Pattern Display* option, in the order of full, tracks, boundary, and periods.

Arguments

w_windowId	The window ID of the window specified.

Value Returned

t_snapPatternDisplayMode

Returns the snap pattern display value.

Example

If the *Snap Pattern Display* option is currently set to `full`, the snap pattern display value will be cycled as follows:

```
leCycleSnapPatternDisplay() -> "tracks"
leCycleSnapPatternDisplay() -> "boundary"
leCycleSnapPatternDisplay() -> "periods"
leCycleSnapPatternDisplay() -> "full"
leCycleSnapPatternDisplay() -> "tracks"
```

leGetGlobalGridsVisible

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

Description

Returns the visibility setting of global snap pattern grids.

Arguments

None

Value Returned

`t`	The global snap pattern grid is visible.
`nil`	The global snap pattern grid is not visible.

Example

Returns `t` if the global snap pattern grid is visible.

```
leGetGlobalGridsVisible()
```

leGetLocalGridsVisible

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

Description

Returns the visibility setting of local snap pattern grids.

Arguments

None

Value Returned

`t`	The local snap pattern grid is visible.
`nil`	The local snap pattern grid is not visible.

Example

Returns `t` if the local snap pattern grid is visible.

```
leGetLocalGridsVisible()
```

leGetSnapPatternVisible

```
leGetSnapPatternVisible(
     t_spDefName
     )
     => t / nil
```

Description

Returns the visibility setting of snap pattern grid with the specified name.

Arguments

t_spDefName	The name of the snap pattern grid.

Value Returned

`t`	The snap pattern grid is visible.
`nil`	The snap pattern grid is not visible.

Example

Returns `t` if the snap pattern grid `spDef1` is visible.

```
leGetSnapPatternVisible("spDef1")
```

leGetSnapToSPTransform

```
leGetSnapToSPTransform(
     figId
     [ ?snapPatternDepth n_snapPatternDepth ]
     [ ?movingItemDepth n_movingItemDepth ]
     [ ?snapPatternLPP g_snapPatternLPP ]
     )
     => x_InstTransform
```

Description

Returns the transform required to align the given figure with an underlying snap pattern or global grid. If a transform is not found, the function returns the identity transform.

Arguments

figId
: The figure ID of the input figure. The valid figure types are instances, rectangles, polygons, vias, and figGroups.

?snapPatternDepth *n_snapPatternDepth*
: The hierarchy depth to be used when looking for snap patterns in fixed instances. The default is `0`.

?movingItemDepth *n_movingItemDepth*
: The hierarchy depth to be used when looking for snap patterns or active shapes while moving instances. The default is `1`.

?snapPatternLPP *g_snapPatternLPP*
: The LPP of the snap pattern that is active for snapping. If a value is provided, then snap patterns on other LPPs are ignored for snapping. The default is `nil`, which implies that all snap patterns are active.

Value Returned

x_InstTransform	Returns the transform required to align the given figure ID with an underlying snap pattern or global grid. If a transform is not found, then the function returns the identity transform.

Example

```
leGetSnapToSPTransform(fig)
leGetSnapToSPTransform(fig ?snapPatternLPP "FinArea type3")
leGetSnapToSPTransform(fig ?snapPatternDepth 1)
leGetSnapToSPTransform(css() ?movingItemDepth 0)
```

leGetWidthSpacingGridsVisible

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

Description

Returns the visibility setting of width spacing snap pattern grids.

Arguments

None

Value Returned

`t`	The width spacing grids are visible.
`nil`	The width spacing grids are not visible.

Example

Returns `t` if the width spacing snap pattern grids are visible.

```
leGetWidthSpacingGridsVisible()
```

leGetWidthSpacingSnapPatternVisible

```
leGetWidthSpacingSnapPatternVisible(
     t_wspName
     )
     => t / nil
```

Description

Returns the visibility setting of the specified width spacing snap pattern grid.

Arguments

t_wspName	The name of the width spacing snap pattern grid, as displayed in the Palette.

Value Returned

`t`	The specified grid is visible.
`nil`	The specified grid is not visible.

Example

Returns the visibility setting of the width spacing snap pattern grid `W1`.

```
leGetWidthSpacingSnapPatternVisible("W1")
=> t
```

leSetGlobalGridsVisible

```
leSetGlobalGridsVisible(
     g_isVisible
     )
     => t / nil
```

Description

Turns on or off the visibility of global snap pattern grids.

Arguments

g_isVisible	Boolean specifying whether the global snap pattern grid is visible. Valid Values: `t` for visible or `nil` for invisible

Value Returned

`t`	The global snap pattern grid is visible.
`nil`	The global snap pattern grid is not visible.

Example

Returns `t` if global snap pattern grid is set to visible and `nil` if it is not.

```
leSetGlobalGridsVisible(t)
```

leSetLocalGridsVisible

```
leSetLocalGridsVisible(
     g_isVisible
     )
     => t / nil
```

Description

Turns on or off the visibility of local snap pattern grids.

Arguments

g_isVisible	Boolean specifying whether the local snap pattern grid is visible. Valid Values: `t` for visible or `nil` for invisible

Value Returned

`t`	The local snap pattern grid is visible.
`nil`	The local snap pattern grid is not visible.

Example

Returns `t` if local snap pattern grid is set to visible and `nil` if it is not.

```
leSetLocalGridsVisible(t)
```

leSetSnapPatternVisible

```
leSetSnapPatternVisible(
     t_spDefName
     g_isVisible
     )
     => t / nil
```

Description

Turns on or off the visibility of the specified snap pattern grid.

Arguments

t_spDefName	The name of the snap pattern grid.
g_isVisible	Boolean specifying whether the snap pattern grid is visible. Valid Values: `t` for visible or `nil` for invisible

Value Returned

`t`	The snap pattern grid is visible.
`nil`	The snap pattern grid is not visible.

Example

Returns `t` if the snap pattern grid `spDef1` is set to visible.

```
leSetSnapPatternVisible("spDef1" t)
```

leSetWidthSpacingSnapPatternVisible

```
leSetWidthSpacingSnapPatternVisible (
     t_gridName
     g_isVisible
     )
     => t / nil
```

Description

Turns on or off the visibility of the specified width spacing snap pattern grid.

Arguments

t_gridName	The name of the width spacing snap pattern grid.
g_isVisible	Boolean specifying whether the width spacing snap pattern grid is visible. Valid Values: `t` for visible or `nil` for invisible.

Value Returned

`t`	The width spacing snap pattern grid visibility setting could be set successfully.
`nil`	The width spacing snap pattern grid visibility setting could not be set successfully.

Example

Turns on the visibility of `W1` width spacing snap pattern grid:

```
leSetWidthSpacingSnapPatternVisible("W1" t)
=> t
```

Turns off the visibility of `W1` width spacing snap pattern grid:

```
leSetWidthSpacingSnapPatternVisible("W1" nil)
=> t
```

The visibility setting could not be applied:

```
leSetWidthSpacingSnapPatternVisible("W3" nil)
=> nil
```

leToggleAllGravity

```
leToggleAllGravity(
     )
     => t
```

Description

Toggles the gravity and rule gravity settings. If the values of gravity (cdsenv `gravityOn`) and rule gravity (cdsenv `ruleGravityOn`) are different, `leToggleAllGravity()` turns on the value of both the environment variables to make them equal.

Arguments

None

Value Returned

`t`	Gravity and rule gravity are toggled.

Example

If gravity setting cdsenv `gravityOn` is set to `t` and rule gravity setting cdsenv `ruleGravityOn` is set to `nil`, the values when invoking the `leToggleAllGravity()` sequentially are listed as follows:

```
leToggleAllGravity()

gravityOn = t; ruleGravityOn = t
```

In case of different value for `gravityOn` and `ruleGravityOn`, the function sets the value variables to `t`.

```
leToggleAllGravity()

gravityOn = nil, ruleGravityOn = nil
leToggleAllGravity()

gravityOn = t, ruleGravityOn = t
```

wspCheckActive

```
wspCheckActive(
     d_cellviewId
     t_layerName | ?shapes g_selSetId
     [ ?region l_bBox ]
     [ ?purpose t_purposeName ]
     [ ?hilightSet g_hlSetId ]
     [ ?markers g_markers [ ?vioLimit g_limit ] ]
     [ ?checkColor g_checkColor ]
     [ ?checkWidth g_checkWidth ]
     [ ?checkWireType g_checkWireType ]
     [ ?depth x_depth ]
     [ ?mergeShapes g_mergeShapes [ ?ignoreColorOnMerge g_ignoreColor ] ]
     [ ?mergePurposes l_mergePurposes ]
     [ ?returnVioShapes g_returnVioShapes ]
     [ ?excludeBlockageCheck g_excludeBlockageCheck ]
     [ ?ignoreValidJogs g_ignoreValidJogs ]
     )
     => t / l_dbID / nil
```

Description

Checks shapes and blockages for width, color, and wireType conformance to the active width spacing pattern (WSP) on the specified layer or in the specified set. The active WSP can be for the global grid or a pattern region.

Arguments

d_cellViewId	Database ID of the cellview to be checked.
t_layerName	Name of the layer to be checked. This argument must be a valid layer name but will be ignored if `?shapes` is specified.
`?shapes` *g_selSetId*	Database ID of the set of shapes to be checked. If the set is empty or contains objects that are not shapes, an error message is displayed and no checking is done.

`?region` *`l_bBox`*
: (Applies only when `?shapes` is not specified) Area of the cellview to be checked. Specifies a pair of coordinates to denote the lower-left and upper-right corners of the bounding box, such as:
`wspCheckActive(cv ?region list(0:0 1:1))`

If this argument is not specified, the entire cellview bounding box is considered by default. The bounding box is in user-specified units.

`?purpose` *`t_purposeName`*
: (Applies only when `?shapes` is not specified) Name of the purpose to be checked.

If this argument is not specified, only the shapes on the `drawing` purpose for the layer are checked.

`?hilightSet` *`g_hlSetId`*
: Database ID of the highlight set to which the shapes that do not conform to the active WSP color, width, or wireType will be added. If this argument is not specified, non-conforming shapes will not be added to a highlight set.

`?markers` *`g_markers`*
: Specifies whether annotation markers will be created for shapes that do not conform to the active WSP.

Valid values: `t` or `nil` (default).

Note: When this argument is set to `t`, the markers are added to the Annotation Browser without clearing any existing markers.

To limit the number of markers that are created, include the `?vioLimit` argument. You can view the markers in the *DRC/DFM* tab of the Annotation Browser. For more information, see Finding Violations.

`?checkColor` *`g_checkColor`*

Specifies whether the color of each shape will be checked for conformance to the track colors of the active WSP.

Valid values: `t` (default) or `nil`.

Conforming shapes:

- A colored shape on a same color track
- A colored shape on a gray (uncolored) track
- A gray shape on a gray track.

Non-conforming shapes:

- A gray shape on a colored track

`?checkWidth` *`g_checkWidth`*

Specifies whether the width of the shape must match the width of the track.

Valid values: `t` (default) or `nil`.

If set to `nil`, the width of the shape does not need to match the width of the track. However, the shape is conforming only if its centerline is aligned with the centerline of the track. Use this setting to check against zero-width tracks.

`?checkWireType` *`g_checkWireType`*

Specifies whether the wireType of each shape must conform to the wireType of track for the active WSP.

Valid values: `t` (default) or `nil`.

Conforming shapes:

- A shape with no wireType on a track with no wireType
- A shape with no wireType on a track with a wireType
- A shape with a wireType on a track with the same wireType

Non-conforming shapes:

- A shape with a wireType on a track with no wireType
- A shape with a wireType on a track with a different wireType

`?depth` *`x_depth`*
: Specifies the hierarchy depth up to which the cellview will be checked.

  The default is `32`.

`?mergeShapes` *`g_mergeShapes`*
: Specifies whether to merge same-color shapes and uncolored shapes that are on the same layer-purpose pair before checking for WSP conformance. When set to `t`, false violations on small shapes (for example, shapes overlapped by a WSP-conforming power rail) can be avoided.

  Valid values: `t` or `nil` (default).

`?mergePurposes` *`l_mergePurposes`*
: Merges the shapes on the purposes in the list that are on the same layer and have the same color before checking for WSP conformance.

`?vioLimit` *`g_limit`*
: Specifies the maximum number of markers to be created. You must specify `?markers t`, or this argument will be ignored.

  The default is `1000`.

`?ignoreColorOnMerge` *`g_ignoreColor`*
: (Applies only when `?checkColor` is `nil` and `?mergeShapes` is `t`) Specifies whether to merge shapes on the same layer-purpose pair, regardless of color, before checking for WSP conformance. When set to `nil`, only the shapes on the same layer-purpose pair with the same color or no coloring are merged.

  Valid values: `t` or `nil` (default)

`?returnVioShapes` *`g_returnVioShapes`*
: Specifies whether to return the list of shapes that do not conform to the active WSP. When set to `t`, the function returns list of database IDs of shapes that do not conform to the active WSP.

  Valid values: `t` or `nil` (default).

`?excludeBlockageCheck` *`g_excludeBlockageCheck`*
: Specifies whether blockage objects will be checked for snap patterno or width spacing pattern conformance.

  Valid values: `t` or `nil` (default).

`?ignoreValidJogs` *`g_ignoreValidJogs`*

Specifies whether to ignore jogs that are defined in the `allowedWidthRanges` constraint for the layer in the jog direction before checking.

Valid values: `t` or `nil` (default).

Value Returned

`t`	All the shapes conform to the active WSP.
`l_dbID`	(Applies only when `?returnVioShapes t` is specified). List of database IDs for shapes that do not conform to the active WSP.
`nil`	If `?returnVioShapes t` has been specified, all the shapes conform to the active WSP. If `?returnVioShapes t` has not been specified, some shapes do not conform to the active WSP.

Examples

The following example checks the `Metal1:drawing` shapes in the edit cellview for width and color conformance to the active WSP. Only the top-level `Metal1:drawing` shapes in the region bounded by (2:1 4:2) are checked.

```
wspCheckActive( geGetEditCellView() "Metal1" ?markers t ?checkWireType nil ?depth
0 ?region list(2:1 4:2) )
```

The following example checks the `Metal2:drawing` shapes in the edit cellview for width, color, and wireType conformance to the active WSP. Non-conforming shapes are added to the `hset` highlight set and are drawn with a halo in the canvas.

```
hset = geCreateWindowHilightSet( hiGetCurrentWindow() list("Metal2" "drawing") )
geSetHilightSetHaloParameters( hset "under" "fadeout" "thick" 33 nil )
retval = wspCheckActive( geGetEditCellView() "Metal2" ?hilightSet hset )
```

The following example checks the shapes in the selection set for width, color, and wireType conformance to the active WSP. Markers are created for the non-conforming shapes.

```
myset = geGetSelSet()
retval = wspCheckActive( geGetEditCellView() ?shapes myset ?markers t )
```

The following example checks the `Metal1:drawing` shapes for centerline alignment, color, and wireType conformance to the active WSP. Markers are created for the non-conforming shapes.

```
wspCheckActive( geGetEditCellView() "Metal1" ?checkWidth nil ?markers t )
```

The following example merges the same-color shapes on the `drw1` and `drw2` purposes of `Metal1` before checking for WSP conformance. For example, all the touching `mask1Color` shapes on `Metal1:drw1` and `Metal1:drw2` will be merged, and all the touching uncolored shapes on `Metal1:drw1` and `Metal1:drw2` will be merged.

```
wspCheckActive( geGetEditCellView() "Metal1" ?mergePurposes list("drw1" "drw2") )
```

The following example merges `Metal1:drawing` shapes, then checks them for centerline alignment, width, and wireType conformance to the active WSP.

```
wspCheckActive( geGetEditCellView() "Metal1" ?checkColor nil ?mergeShapes t
?ignoreColorOnMerge t)
```

Related Topics

Space-based Router Batch Checking (WSP Active Checking)

wspCreateWSP

```
wspCreateWSP (
     d_cellViewId
     t_name
     [ ?offset n_offset ]
     [ ?repeatOffset g_repeatOffset ]
     [ ?shiftColor g_shiftColor ]
     [ ?allowedRepeatMode t_allowedRepeatMode ]
     [ ?defaultRepeatMode t_defaultRepeatMode ]
     [ ?wsspDef t_wsspDefName ]
     n_width n_space [ repeat n_repeats ] [ wireType t_wireType ] [ color n_mask
     ] [ displayPacket t_displayPacket ]
     )
     => d_wspId
```

Description

Creates a width spacing pattern (WSP) in the specified cellview. If the WSP already exists, it is replaced. This function creates simple WSPs and not nested lists. It checks the shapes for width, color, and wire type conformance to the active WSP on the specified layer or set. The active WSP can be for the global grid or a pattern region.

Arguments

d_cellViewId	Database ID of the cellview in which the WSP is created.
t_name	Name of the WSP.
?offset *n_offset*	Center-line offset of the first track. The default is `0.0`.
?repeatOffset *g_repeatOffset*	The offset applied each time the pattern repeats. The default is `t`.
?shiftColor *g_shiftColor*	The track color shift when the pattern repeats. The default is `nil`.
?allowedRepeatMode *t_allowedRepeatMode*	Specifies the way the pattern can repeat. The choices are `any`, `none`, `steppedOnly`, or `flippedOnly`. The default is `any`.
?defaultRepeatMode *t_defaultRepeatMode*	

	Specifies the default repeat mode when the `allowedRepeatMode` is specified. The choices are `noRepeat`, `stepped`, `flippedStartsWithOdd`, or `flippedStartsWithEven`. The default is `noRepeat`.
?wsspDef t_wsspDefName	
	Specifies the pattern as allowed for the WSSPDef. If the WSSPDef was previously created without a default active, its creation is deferred and it is created along with this WSP.
n_width	Specifies the width of the track.
n_space	Specifies the center-line space to the next track.
n_repeats	Specifies the number of times the track repeats. The default is `1`.
t_wireType	Specifies the type of wire.
t_color	Specifies the color of the track. The valid values are `1`, `2`, and `3`.
t_displayPacket	Specifies the display packet name to be used for displaying the specified WSP track.

Value Returned

d_wspId	Database ID of the WSP.

Examples

```
wspCreateWSP(cv
             "test"
             ?offset 0.01
             ?shiftColor t
             ?allowedRepeatMode "steppedOnly"
             ?defaultRepeatMode "stepped"
             0.030  0.060
             0.031  0.061  "repeat:2"
             0.032  0.062   "color:1"
             0.033  0.063   "wireType:vss1" "color:2"
             0.034  0.064  "color:1"   "displayPacket:myCustomTrackPacket"
    )
```

wspCreateWSPByAttr

```
wspCreateWSPByAttr (
     d_cellViewId
     list (nil
          'name t_name
          'offset n_offset
          'shiftColor g_shiftColor
          'repeatOffset g_repeatOffset
          'allowedRepeatMode t_allowedRepeatMode
          'defaultRepeatMode t_defaultRepeatMode
          tracks  ( list(nil
                    'width n_width
                    'space n_space
                    [ 'wireType t_wireType ]
                    [ 'color t_color ]
                    [ 'displayPacket t_displayPacket ]
                     )
     )
     )
     => d_wspId
```

Description

Creates a width spacing pattern (WSP). The pattern attributes are specified as a DPL. This function is specified with the `wspGetFlatAttr` function, which returns an existing pattern as a DPL. It lets you convert the existing patterns to a DPL, modify them, and recreate them.

Arguments

d_cellViewId	Database ID of the cellview in which the WSP is created.
'name *t_name*	Name of the WSP.
'offset *n_offset*	Center-line offset of the first track. The default is `0.0`.
'shiftColor *g_shiftColor*	The track color shift when the pattern repeats. The default is `nil`.
'repeatOffset *g_repeatOffset*	The offset applied each time the pattern repeats. The default is `t`.
'allowedRepeatMode *t_allowedRepeatMode*	

Specifies the way the pattern can repeat. The choices are `any`, `none`, `steppedOnly`, or `flippedOnly`. The default is `any`.

`'defaultRepeatMode` *`t_defaultRepeatMode`*

Specifies the default repeat mode, when the `allowedRepeatMode` is specified. The choices are `noRepeat`, `stepped`, `flippedStartsWithOdd`, or `flippedStartsWithEven`. The default is `noRepeat`.

`'width` *`n_width`* Specifies the width of the track in user units.

`'space` *`n_space`* Specifies the center-line space to the next track.

`'wireType` *`n_wireType`*`'`

Specifies the wire type.

`'color` *`t_color`* Specifies the color of the track. The values are `mask1Color`, `mask2Color`, or `mask3Color`.

`'displayPacket` *`t_displayPacket`*

Specifies the display packet name to be used for displaying the specified WSP track.

Value Returned

`d_wspId` Database ID of the WSP.

Examples

```
wspCreateWSPByAttr(cv list (nil 'name "mflat' 'shiftColor t 'allowedRepeatMode
"any" 'defaultRepeatMode "stepped" 'tracks list(
list(nil 'width 0.064 'space 0.096 "maskColor")
list(nil 'width 0.032 'space 0.064 'color "mask2Color" 'wireType "new")
list(nil 'width 0.032 'space 0.064 'color "mask1Color" 'wireType "new"
'displayPacket "m2")
)
'repeatOffset t 'offset 0.01 )
)
```

wspCreateWSPGroup

```
wspCreateWSPGroup (
     d_cellViewId
     t_name
     l_members
     )
     => d_wspGroupId
```

Description

Creates a width spacing pattern (WSP) group in the specified cellview. The argument after *t_name* are interpreted as group members.

Arguments

d_cellViewId	Database ID of the cellview in which the WSP is created.
t_name	Name of the WSP to be created.
l_members	List of WSP names to be added to the group.

Value Returned

d_wspGroupId	Database ID of the WSP group.

Example

Creates `group1X` and the WSPs `wsp_1X1X` and `wsp_1X_gap_1X` are added to it.

```
wspCreateWSPGroup(cv "group1X" list( "wsp_1X1X" "wsp_1X_gap_1X") )
```

wspCreateWSSPDef

```
wspCreateWSSPDef (
     d_cellViewId
     t_name
     tx_layer
     t_direction
     n_period
     [ ?defaultActive t_defaultActive ]
     [ ?allowedPatterns l_patterns ]
     [ ?allowedPatternGroups l_groups ]
     [ ?defPurpose tx_purpose ]
     [ ?offset n_offset ]
     [ ?orthogonalGrid t_orthogonalGrid]
     )
     => d_wsspDefId
```

Description

Creates a `widthSpacingSnapPatternDefs` (WSSPDef) in the specified cellview. If *`t_defaultActive`* is specified, the WSSPDef is created immediately. If it is not specified, the creation of WSSPDef is deferred until the first call to the function `wspCreateWSP` that references this WSSPDef. This enables all WSSPDefs to be specified first, followed by calls to `wspCreateWSP` that reference the WSSPDef.

Arguments

`d_cellViewId`	Database ID of the cellview in which the WSSPDef is created.
`t_name`	Name of the WSSPDef.
`tx_layer`	The routing layer to which the WSSPDef applies. This is also used as the layer in which the WSP regions for this WSSPDef are drawn.
`t_direction`	The direction in which track spacing is measured. This is orthogonal to the routing direction. The choices are `vertical` or `horizontal`.
`n_period`	The period of the WSSPDef.
`?defaultActive` *`t_defaultActive`*	The name of the default WSP to use.
`?allowedPatterns` *`l_patterns`*	List of the allowed WSPs. If not specified, the defaultActive pattern is the only allowed pattern.

`?allowedPatternGroups` *`l_groups`*

	List of the allowed WSP groups.
`?defPurpose` *`tx_purpose`*	The purpose to use for WSP regions. The default purpose is `track`.
`?offset` *`n_offset`*	The global grid offset.
`?orthogonalGrid` *`t_orthogonalGrid`*	The orthogonal grid. The choices are `upperLower`, `upper`, or `lower`.

Value Returned

`d_wsspDefId`	Database ID of the WSSPDef.

Example

```
wspCreateWSSPDef(geGetEditCellView() "M1WSP1" "Metal1" "Vertical" 0.45)
```

wspCreateWSSPDefByAttr

```
wspCreateWSSPDefByAttr (
     d_cellViewId
     l_attrs
     )
     => d_wspId
```

Description

Creates a `widthSpacingSnapPatternDefs` (WSSPDef) using an attribute DPL.

Arguments

d_cellViewId	Database ID of the cellview.
l_attrs	List of attributes specified in the DPL. The list of attributes contains the `orthogonalGrid`.

Value Returned

d_wspId	Database ID of the WSSPDef.

Example

```
attrlist = (nil defaultActiveName "minWidth" patternGroupNames nil
patternNames nil snappingLayers table:snapping layer table look-up offset
0.0 direction "vertical" period 0.768
purpose "localWSP" layer "Metal5" name
"snpLppsTest"
)
wspCreateWSSPDefByAttr(cvId attrlist)
```

wspDeleteCellViewAllWSPData

```
wspDeleteCellViewAllWSPData (
     d_cellViewId
     )
     => t / nil
```

Description

Deletes all the width spacing pattern (WSP) data from the specified cellview.

Arguments

d_cellViewId	Database ID of the cellview for which all the WSP data is to be deleted.

Value Returned

`t`	The WSP data is deleted.
`nil`	The WSP data is not deleted.

Example

```
wspDeleteCellViewAllWSPData(geGetEditCellView())
```

wspDeleteMetalFill

```
wspDeleteMetalFill (
     d_cellViewId
     t_LayerName
     ?removeTrim g_removeTrim
     ?removePatch g_removePatch
     ?removePassive g_removePassive
     )
     => t / nil
```

Description

Deletes the shapes created by the `wspMetalFillTrim()` function.

Arguments

d_cellViewId	Database ID of the cellview for which the shapes are to be deleted.
t_LayerName	Name of the layer in which filled shapes are searched.
`?removeTrim` *g_removeTrim*	Deletes filled trim shapes only. The default is `nil`.
`?removePatch` *g_removePatch*	Deletes filled patch shapes only. The default is `nil`.
`?removePassive` *g_removePassive*	Deletes filled passive shapes only. The default is `nil`.

Value Returned

`t`	The command executed successfully.
`nil`	The command did not execute successfully.

Example

```
cv = geGetEditCellView()
```

```
wspDeleteMetalFill(cv "Metal2"  ?region cv~>bBox ?removePassive t)
```

wspDumpToFile

```
wspDumpToFile (
     d_cellViewId
     t_fileName
     )
     => t / nil
```

Description

Creates a SKILL file that can be used to recreate the width spacing pattern (WSP) information in the specified cellview.

Arguments

d_cellViewId	Database ID of the cellview.
t_fileName	The name of the SKILL file to which the WSP information is written.

Value Returned

`t`	The SKILL file is created.
`nil`	The SKILL file is not created.

Example

```
wspDumpToFile(geGetEditCellView() "./cvNameWSPs.il")
```

wspGetLineEndGrids

```
wspGetLineEndGrids (
     d_cellViewId
     t_layerName
     )
     => l_gridNames / nil
```

Description

Returns a list of line-end grid names for the specified layer.

Arguments

d_cellViewId	Database ID of the cellview.
t_layerName	Name of the layer for which line-end grid names are to be searched.

Value Returned

l_gridNames	List of line-end grid names for the specified layer.
nil	The command did not return line-end grid names.

Example

Returns the line-end grid names for the layer, m1, in the current cellview.

```
wspGetLineEndGrids(geGetEditCellView() "m1")
=>("m1_B" "m1_T")
```

wspGetWSSPDefLP

```
wspGetWSSPDefLP(
     d_wsspDefId
     )
     => t_layerName t_purposeName
```

Description

Returns the layer and purpose name of a widthSpacingSnapPatternDef.

Arguments

d_wsspDefId	The database ID of the WSSPDef to be queried.

Value Returned

t_layerName	Name of the layer associated with the WSSPDef.
t_purposeName	Name of the purpose associated with the WSSPDef.

Example

```
wspGetWSSPDefLP(wsp1)
```

wspRegionFindByLayer

```
wspRegionFindByLayer (
     d_cellViewId
     t_routingLayerName
     )
     => l_regionIds
```

Description

Returns a list of the width spacing pattern (WSP) region IDs for the specified routing layer name in a cellview.

Arguments

d_cellViewId	Database ID of the cellview.
t_routingLayerName	Name of the routing layer.

Value Returned

l_regionIds	List of WSP region IDs.

Example

```
wspRegionFindByLayer(geGetEditCellView() "Metal2")
```

wspRegionFindByWSSPDef

```
wspRegionFindByWSSPDef (
     d_cellViewId
     t_WSSPDefName
     )
     => l_regionIds
```

Description

Returns a list of region IDs in the specified cellview for a widthSpacingSnapPatternDefs (WSSPDef).

Arguments

d_cellViewId	Database ID of the cellview.
t_WSSPDefName	Name of the WSSPDef.

Value Returned

l_regionIds	List of region IDs for the specified WSSPDef.

Example

```
wspRegionFindByWSSPDef(geGetEditCellView() "M1WSPDEFP90")
```

wspRegionGetActivePattern

```
wspRegionGetActivePattern(
     d_regionId
     )
     => t_activePatternName
```

Description

Returns the name of the active pattern for a width spacing pattern (WSP) region.

Arguments

d_regionId	The database ID of the region.

Value Returned

t_activePatternName

The name of the active pattern for a WSP region.

Example

```
wspRegionGetActivePattern(regionId)
```

wspRegionGetAllowedPatternGroups

```
wspRegionGetAllowedPatternGroups (
     d_regionId
     )
     => l_allowedPatternGroupNames
```

Description

Returns a list of names of allowed pattern groups for the specified width spacing pattern (WSP) region.

Arguments

d_regionId	The database ID of the WSP region.

Value Returned

l_allowedPatternGroupNames

List of allowed pattern group names for the specified WSP region.

Example

```
wspRegionGetAllowedPatternGroups(regionId)
```

wspRegionGetAllowedPatterns

```
wspRegionGetAllowedPatterns (
     d_regionId
     [ ?excludeGroups g_excludeGroups ]
     )
     => l_allowedPatternNames
```

Description

Returns a list with the names of the allowed patterns for a width spacing pattern (WSP) region. By default, this function returns the effective set of patterns by combining the allowed patterns and allowed pattern groups. You can use the `g_excludeGroups` argument to ignore the allowed pattern groups.

Arguments

d_regionId The database ID of the WSP region.

?excludeGroups *g_excludeGroups*

Ignores allowed pattern groups.

Value Returned

l_allowedPatternNames

List of allowed patterns for a WSP region.

Example

```
wspRegionGetAllowedPatterns(regionId)
```

wspRegionGetWSSPDef

```
wspRegionGetWSSPDef(
     d_regionId
     )
     => d_wsspDefId
```

Description

Returns the ID of the widthSpacingSnapPatternDefs (WSSPDef) to which the specified region is bound.

Arguments

d_regionId	The database ID of the region.

Value Returned

d_wsspDefId	The database ID for a WSSPDef.

Example

```
wspRegionGetWSSPDef(regionId)
```

wspSetWSSPDefRegionPurpose

```
wspSetWSSPDefRegionPurpose (
     d_wsspDefId
     t_purposeName
     )
     => t / nil
```

Description

Overrides the region purpose attribute for the specified widthSpacingSnapPatternDefs (WSSPDef).

Arguments

d_wsspDefId	Database ID of the WSSPDef.
t_purposeName	The region purpose name.

Value Returned

`t`	The region purpose attribute is overridden.
`nil`	The region purpose attribute is not overridden.

Example

```
wspSetWSSPDefRegionPurpose(ldef "localWSP")
```

wspSPDefFind

```
wspSPDefFind (
     d_cellViewId
     t_layerName
     [ ?enabledOnly g_enabledOnly ]
     )
     => l_spdef / nil
```

Description

Finds all the snapPatternDefs (SPDefs) that apply to the specified routing layer name. In a cellview, one SPDef or widthSpacingSnapPatternDefs (WSSPDef) per layer is enabled as a global grid. With the optional argument *g_enabledOnly* set to `t`, WSSPDefs that are enabled as global for the specified routing layer are returned.

Arguments

d_cellViewId	Database ID of the cellview.
t_layerName	The routing layer on which to search for WSSPDefs.
?enabledOnly *g_enabledOnly*	When set to `t`, returns WSSPDefs that are enabled as global. When set to `nil`, returns all WSSPDefs regardless of whether they are enabled as global or not. The default value is `nil`.

Value Returned

l_spdef	List of SPDef IDs.
`nil`	The SPDef IDs are not returned.

Example

Returns a list of all SPDefs that apply on layer `Metal1` and that are enabled as global.

```
wspSPDefFind(cv "Metal1" ?enabledOnly t)
```

wspWSPFindByName

```
wspWSPFindByName (
     d_cellViewId
     t_wspname
     [ cellViewOnly g_cellViewOnly ]
     )
     => d_wspId
```

Description

Finds a width spacing pattern (WSP) by name in a cellview or in the technology database.

Arguments

d_cellViewId	Database ID of the cellview.
t_wspname	The name of the WSP to search for.
g_cellViewOnly	Specifies whether the WSP must be searched only in the cellview (`t`) or first in the cellview and then in the technology database (`nil`).

Value Returned

d_wspId	The database ID of the WSP.

Example

```
wspWSPFindByName(geGetEditCellView() "WSP2")
```

wspWSPGetFlatAttr

```
wspWSPGetFlatAttr (
     d_wspId
     [ ?numMasks n_numMasks ]
     )
     => l_wspAttrs
```

Description

Returns the attributes of a width spacing pattern (WSP) as a DPL.

Arguments

d_wspId	The ID of the WSP for which to get the attribute list.
n_numMasks	The number of masks.

Value Returned

l_wspAttrs	List of attributes of a WSP.

Example

```
wspWSPGetFlatAttr(W1)
```

wspWSPGroupFindByName

```
wspWSPGroupFindByName (
     d_cellViewId
     t_wspname
     [ cellViewOnly g_cellViewOnly ]
     )
     => d_wspGroupId
```

Description

Finds a width spacing pattern (WSP) group by its name in a cellview or in the technology database.

Arguments

d_cellViewId	Database ID of the cellview.
t_wspname	The name of the WSP group to search for.
g_cellViewOnly	Specifies whether the WSP group is searched only in the cellview (`t`) or it is searched first in the cellview and then in the technology database (`nil`).

Value Returned

d_wspGroupId	The database ID of the WSP group.

Example

```
wspWSPGroupFindByName(geGetEditCellView() "WSP2GP1")
```

wspWSSPDefAddToAllowedPatternGroups

```
wspWSSPDefAddToAllowedPatternGroups(
     d_cellViewId
     t_wsspDefName
     t_patternGroupName
     )
     => t / nil
```

Description

Adds a width spacing pattern (WSP) group to the list of allowed pattern groups for a WSSPDef.

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	The name of the WSSPDef.
t_patternGroupName	The name of the pattern group to be added.

Value Returned

t	The WSP is added to the allowed pattern groups for a WSSPDef.
nil	The WSP is not added to the allowed pattern groups for a WSSPDef.

Example

```
wspWSSPDefAddToAllowedPatternGroups(geGetEditCellView() "M1AllPeriod190"
"WSP2RSP")
```

wspWSSPDefAddToAllowedPatterns

```
wspWSSPDefAddToAllowedPatterns(
     d_cellViewId
     t_wsspDefName
     t_patternName
     )
     => t / nil
```

Description

Adds a width spacing pattern (WSP) to the allowed patterns for a widthSpacingSnapPatternDefs (WSSPDef).

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	The name of the WSSPDef.
t_patternName	The name of the pattern.

Value Returned

t	The WSP is added to the allowed patterns for a WSSPDef.
nil	The WSP is not added to the allowed patterns for a WSSPDef.

Example

```
wspWSSPDefAddToAllowedPatterns(geGetEditCellView() "M1AllPeriod190" "WSP2")
```

wspWSSPDefAddToEnabled

```
wspWSSPDefAddToEnabled(
     d_cellViewId
     l_wsspDefName
     )
     => t / nil
```

Description

Adds a list of widthSpacingSnapPatternDefs (WSSPDefs) to the list of WSSPDefs enabled as global grids in the specified cellview. This will filter out any duplicate, undefined wsspDefs, and wsspDefs with direction mis-match.

Arguments

d_cellViewId	Database ID of the cellview.
l_wsspDefName	List of names of WSSPDefs to be added to the list of WSSPDefs.

Value Returned

t	The WSSPDefs added to the list of WSSPDefs.
nil	The WSSPDefs not added to the list of WSSPDefs.

Example

```
wspWSSPDefAddToEnabled(geGetEditCellView() list("M1AllPeriod190"))
```

wspWSSPDefClearEnabledOverrides

```
wspWSSPDefClearEnabledOverrides(
     d_cellViewId
     )
     => t / nil
```

Description

Removes any global grid overrides from the specified cellview.

Arguments

d_cellViewId	Database ID of the cellview.

Value Returned

t	Global grid overrides were removed.
nil	Global grid overrides were not removed.

Example

```
wspWSSPDefClearEnabledOverrides(geGetEditCellView())
```

wspWSSPDefFind

```
wspWSSPDefFind (
     d_cellViewId
     t_layerName
     [ ?enabledOnly g_enabledOnly ]
     )
     => l_wsspDefIds
```

Description

Returns a list of widthSpacingSnapPatternDefs (WSSPDefs) that apply to a given routing layer name. The WSSPDefs may exist in the cellview or in the technology database. With the optional argument *g_enabledOnly* set to `t`, routing WSSPDefs that are enabled as global for the given layer are returned.

Arguments

d_cellViewId	Database ID of the cellview.
t_layerName	Routing layer on which WSSPDefs are searched.
?enabledOnly *g_enabledOnly*	When set to `t`, returns the routing WSSPDefs that are enabled as global. When set to `nil`, returns all WSSPDefs, routing and orthogonal, regardless of whether they are enabled as global. The default value is `nil`.

Value Returned

l_wsspDefIds	List of WSSPDef IDs.

Example

```
wspWSSPDefFind(geGetEditCellView() "M2")
```

wspWSSPDefFindByName

```
wspWSSPDefFindByName (
     d_cellViewId
     t_wspname
     [ ?cellViewOnly g_cellViewOnly ]
     )
     => d_wsspDefId
```

Description

Finds a widthSpacingSnapPatternDefs (WSSPDef) by its name in a cellview or technology database.

Arguments

d_cellViewId	Database ID of the cellview.
t_wspname	The name of the WSSPDef to search for.
g_cellViewOnly	Specifies whether the WSSPDef is searched only in the cellview (`t`) or it is searched first in the cellview and then in the technology database (`nil`).

Value Returned

d_wsspDefId	Database ID of the WSSPDef.

Example

```
wspWSSPDefFindByName(geGetEditCellView() "M1AllPeriod190")
```

wspWSSPDefGetActivePattern

```
wspWSSPDefGetActivePattern(
     d_cellViewId
     d_wsspDefId
     )
     => t_activePatternName
```

Description

Returns the active pattern name for a widthSpacingSnapPatternDefs (WSSPDef) in the specified cellview. The active pattern may be set by a constraint in the cellview or from the WSSPDef in the cellview or technology database.

Arguments

d_cellViewId	Database ID of the cellview.
d_wsspDefId	The database ID of the WSSPDef.

Value Returned

t_activePatternName

The active pattern name for the WSSPDef.

Example

```
wspWSSPDefGetActivePattern(geGetEditCellView() wsspDefId)
```

wspWSSPDefGetAllowedPatternGroups

```
wspWSSPDefGetAllowedPatternGroups(
     d_cellViewId
     d_wsspDefId

     )
     => l_allowedPatternGroupNames
```

Description

Returns a list with the names of the allowed pattern groups for the specified widthSpacingSnapPatternDefs (WSSPDef).

Arguments

d_cellViewId	Database ID of the cellview.
d_wsspDefId	Database ID of the WSSPDef.

Value Returned

l_allowedPatternGroupNames

The list of the allowed pattern group names for the WSSPDef.

Example

```
wspWSSPDefGetAllowedPatternGroups(geGetEditCellView() wsspDefId)
```

wspWSSPDefGetAllowedPatterns

```
wspWSSPDefGetAllowedPatterns(
     d_cellViewId
     d_wsspDefId
     [ ?excludeGroups g_excludeGroups ]
     )
     => l_allowedPatternNames
```

Description

Returns a list of names of allowed width spacing patterns (WSPs) for a given widthSpacingSnapPatternDefs (WSSPDef) in the specified cellview. A WSSPDef has a list of allowed patterns and allowed pattern groups. The effective set of allowed patterns is a combination of the allowed WSPs and a WSP that is a member of an allowed pattern group. By default, this function returns the effective set of allowed patterns. You can use the `g_excludeGroups` argument to get only the directly allowed WSPs.

Arguments

d_cellViewId	Database ID of the cellview.
d_wsspDefId	The database ID of the WSSPDef.
g_excludeGroups	Ignores allowed pattern groups.

Value Returned

l_allowedPatternNames

List of allowed pattern names for a WSSPDef.

Example

```
wspWSSPDefGetAllowedPatterns(geGetEditCellView() wsspdefId)
```

wspWSSPDefGetAttr

```
wspWSSPDefGetAttr (
     d_wsspDefId
     )
     => l_wsspDefAttrs
```

Description

The attributes of the specified widthSpacingSnapPatternDefs (WSSPDef) are converted to a DPL format. All the attribute representations are the same as the source representation, except the `snappingLayers` attribute, which is stored as an association table for layer-purpose pair lookup.

Arguments

d_wsspDefId	Database ID of the WSSPDef.

Value Returned

l_wsspDefAttrs	The DPL format of the WSSPDef attributes. The list of attributes contains the `orthogonalGrid`.

Example

```
nattrlist = wspWSSPDefGetAttr(ldef)
     => (nil defaultActiveName "minWidth" patternGroupNames nil
     patternNames nil snappingLayers table:snapping layer table look-up offset
     0.0 direction "vertical" period 0.768 purpose "localWSP" layer "Metal5" name
     "snpLppsTest")
```

wspWSSPDefGetEnabledOverrides

```
wspWSSPDefGetEnabledOverrides (
     d_cellViewId
     [ ?direction t_direction ]
     )
     => l_wsspDefNames
```

Description

Returns the names of all the widthSpacingSnapPatternDefs (WSSPDefs) in the specified direction that are enabled as global grids. The WSSPDef direction is orthogonal to the routing direction. This filters out any duplicates, undefined wsspDefs, and wsspDefs with a direction mis-match.

Arguments

d_cellViewId	Database ID of the cellview.
t_direction	Direction of WSSPDef. The direction can be `vertical` or `horizontal`.

Value Returned

l_wsspDefNames	List of the names of all WSSPDefs in the specified direction that are enabled as global grids.

Example

```
wspWSSPDefGetEnabledOverrides(geGetEditCellView())
```

wspWSSPDefRemoveFromAllowedPatternGroups

```
wspWSSPDefRemoveFromAllowedPatternGroups(
     d_cellViewId
     t_wsspDefName
     t_patternGroupName
     )
     => t / nil
```

Description

Removes a pattern group from the allowed pattern groups of a widthSpacingSnapPatternDefs (WSSPDef).

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	Name of the WSSPDef.
t_patternGroupName	Name of the pattern group.

Value Returned

t	The pattern group has been removed from the allowed pattern groups of a WSSPDef.
nil	The pattern group has not been removed from the allowed pattern groups of a WSSPDef.

Example

```
wspWSSPDefRemoveFromAllowedPatternGroups(geGetEditCellView() "M1AllPeriod190"
"GP1")
```

wspWSSPDefRemoveFromAllowedPatterns

```
wspWSSPDefRemoveFromAllowedPatterns(
     d_cellViewId
     t_wsspDefName
     t_patternName

     => t / nil
```

Description

Removes a width spacing pattern (WSP) from the allowed patterns of the specified cellview. This function only removes the directly-allowed WSP. If a pattern is allowed because it is a member of an allowed pattern group, then no update is made.

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	Name of the WSSPDef.
t_patternName	Name of the pattern.

Value Returned

t	The WSP has been removed from the allowed patterns.
nil	The WSP has not been removed from the allowed patterns.

Example

```
wspWSSPDefRemoveFromAllowedPatterns(geGetEditCellView() "M1AllPeriod190" "WSP2")
```

wspWSSPDefRemoveFromEnabled

```
wspWSSPDefRemoveFromEnabled(
     d_cellViewId
     l_wsspDefName
     )
     => t / nil
```

Description

Removes the specified widthSpacingSnapPatternDefs (WSSPDefs) from the list of WSSPDefs that are enabled as global grids in the cellview.

If this is the only global WSSPDef in this direction, there will be no global grid in the direction even if the technology database specifies global WSSPDefs in the direction. This function filters duplicates, undefined wsspDefs, and wsspDefs with a direction mismatch.

Arguments

d_cellViewId	Database ID of the cellview.
l_wsspDefName	List of names of the WSSPDefs to be removed from the list of WSSPDefs.

Value Returned

t	The WSSPDefs are removed from the list of WSSPDefs.
nil	The WSSPDefs are not removed from the list of WSSPDefs.

Example

```
wspWSSPDefRemoveFromEnabled(geGetEditCellView() list("WSP2" WSP1"))
```

wspWSSPDefRename

```
wspWSSPDefRename (
     d_wsspDef
     t_newName
     )
     => d_wsspDefId
```

Description

Renames a widthSpacingSnapPatternDefs (WSSPDef) by deleting and re-creating it. So, the DB ID may change. This function updates the global grid and any regions that reference the WSSPDef.

Arguments

d_wsspDef	Database ID of the WSSPDef.
t_newName	The new name of the WSSPDef.

Value Returned

d_wsspDefId	Database ID of the renamed WSSPDef.

Example

```
wspWSSPDefRename(wsspDefId "NewWSPName")
```

wspWSSPDefSetActivePattern

```
wspWSSPDefSetActivePattern(
     d_cellViewId
     t_wsspDefName
     t_defaultActiveName
     )
     => t / nil
```

Description

Sets the default active width spacing pattern (WSP) for a widthSpacingSnapPatternDefs (WSSPDef) in the specified cellview.

Depending on whether the WSSPDef exists in the technology database or cellview, the default active WSP is set in the WSSPDef or a constraint in the cellview.

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	The name of the WSSPDef.
t_defaultActiveName	The name of the default active WSP for the WSSPDef.

Value Returned

t	The default active WSP is set for the WSSPDef.
nil	The default active WSP is not set for the WSSPDef.

Example

```
wspWSSPDefSetActivePattern(geGetEditCellView() "M1_H_160_960" "M1_960_WSP2"
"M1_960_WSP3")
```

wspWSSPDefSetAllowedPatternGroups

```
wspWSSPDefSetAllowedPatternGroups(
     d_cellViewId
     t_wsspDefName
     l_patternGroupNames
     )
     => t / nil
```

Description

Sets the allowed pattern groups for a widthSpacingSnapPatternDefs (WSSPDef) in the specified cellview.

This function does not update the directly allowed width spacing patterns (WSPs). It only updates the allowed pattern groups. To update the directly allowed WSP, use the `wspWSSPDefSetAllowedPatterns` function.

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	The name of the WSSPDef.
l_patternGroupNames	The name of the allowed pattern groups.

Value Returned

`t`	The allowed pattern groups have been updated.
`nil`	The allowed pattern groups have not been updated.

Example

```
wspWSSPDefSetAllowedPatternGroups(geGetEditCellView() "M1_H_160_960"
list("M1_960_WSP1" "M1_960_WSP2" "M1_960_WSP3"))
```

wspWSSPDefSetAllowedPatterns

```
wspWSSPDefSetAllowedPatterns(
     d_cellViewId
     t_wsspDefName
     l_patternNames
     [ ?keepGroups g_keepGroups ]
     )
     => t / nil
```

Description

Sets the allowed patterns for a widthSpacingSnapPatternDefs (WSSPDef) in the specified cellview. If the WSSPDef is enabled as global, the patterns apply to the global grid of the WSSPDef and to any regions for the WSSPDef for which the allowed patterns are not overridden.

This function sets a list of width spacing pattern (WSP) names as the allowed patterns. By default, this function deletes any allowed pattern groups. So, the set of allowed patterns is identical to the list of pattern names specified in the *l_patternNames* argument. You can set the allowed WSP and keep the allowed pattern groups using the *g_keepGroups* argument.

Arguments

d_cellViewId	Database ID of the cellview.
t_wsspDefName	The name of the WSSPDef.
l_patternNames	List of pattern names.
g_keepGroups	Specifies whether to keep the allowed pattern groups.

Value Returned

t	The allowed pattern is set for the WSSPDef.
nil	The allowed pattern is not set for the WSSPDef.

Example

```
wspWSSPDefSetAllowedPatterns(geGetEditCellView() "M1_H_90_80" list("WSP1"))
```

wspWSSPDefSetEnabledOverrides

```
wspWSSPDefSetEnabledOverrides(
     d_cellViewId
     l_wsspDefNames
     [ ?direction t_direction ]
     )
     => t / nil
```

Description

Overrides the global grid in the specified cellview and by enabling the specified list of widthSpacingSnapPatternDefs (WSSPDefs) as global grids. If the `l_wsspDefNames` argument does not contain any WSSPDef in a direction, there is no global grid for that direction even if the technology database specifies a global grid for that direction.

Arguments

d_cellViewId	Database ID of the cellview.
l_wsspDefNames	List of names of WSSPDefs.
t_direction	Direction of WSSPDef. The direction can be `vertical` or `horizontal`.

Value Returned

`t`	The global grid is overridden.
`nil`	The global grid is not overridden.

Example

```
wspWSSPDefSetEnabledOverrides(geGetEditCellView() list("WSP2"))
```

Track Pattern Assistant Functions

Tracks patterns are uniformly-spaced orthogonal patterns extending across a base array. To manage track patterns, you can use the SKILL functions associated with the Virtuoso Layout Suite Track Pattern Assistant. For example, you can use `tpaSetDefVisible` to set the visibility of the active pattern for a given width spacing snap pattern definition (WSSPDef) or snap pattern definition (SPDef) in the Track Pattern table.

Related Topics

tpaDeselect

tpaSelect

tpaSelectWSSPDef

tpaSetActivePattern

tpaSetAllDefsVisible

tpaSetDefVisible

tpaSetFilterByName

tpaSetFilterByNumber

tpaSetRegionActivePattern

tpaSetRegionDefVisible

tpaSetRegionWireType

tpaSetWireType

tpaToggleDef

tpaToggleLayer

tpaToggleRegionLayer

tpaDeselect

```
tpaDeselect(
     [ l_hierarchyName ]
     )
     => t / nil
```

Description

Deselects the specified items on the Track Pattern Assistant.

Arguments

l_hierarchyName	List of hierarchy names of the item to be deselected. If *l_hierarchyName* is not specified, all items are deselected.

Value Returned

t	The specified items were deselected.
nil	The function was not successful.

Example

Deselects the specified items on the Track Pattern Assistant.

```
tpaDeselect(list("Metal2" "M2WSP")) => t | nil
tpaDeselect() => t
```

tpaSelect

```
tpaSelect(
     [ l_hierarchyName ]
     )
     => t / nil
```

Description

Selects the specified item on the Track Pattern Assistant.

Arguments

l_hierarchyName	List of hierarchy names of the item to be selected.

Value Returned

t	The specified items were selected.
nil	The function was not successful.

Example

Selects the specified items in the Track Pattern Assistant.

```
tpaSelect('("Metal2" "M2WSP"))
tpaSelect(list("Local Regions" "FG__0" "Metal3" "M3WSP"))
```

tpaSelectWSSPDef

```
tpaSelectWSSPDef(
     t_defName
     )
     => t / nil
```

Description

Selects the specified widthSpacingSnapPatternDefs (WSSPDef) on the Track Pattern Assistant. This is similar to selecting the WSSPDef in the Track Pattern Assistant.

Arguments

t_defName	Name of the WSSPDef.

Value Returned

`t`	The WSSPDef is selected.
`nil`	The WSSPDef is not selected.

Example

Selects the `M2WSP` WSSPDef on the Track Pattern Assistant.

```
tpaSelectWSSPDef("M2WSP")
```

tpaSetActivePattern

```
tpaSetActivePattern(
     t_defName
     t_activePatternName
     )
     => t / nil
```

Description

Sets the active pattern for the given width spacing snap pattern definition (WSSPDef).

Arguments

t_defName	Name of the WSSPDef.
t_activePatternName	Assigns the named width spacing pattern (WSP) as the active pattern for the specified WSSPDef.

Value Returned

t	The named WSP is the active pattern for the specified WSSPDef.
nil	The active pattern was not set because the WSP or the WSSPDef was not found.

Example

Sets the `minWidth` WSP as the active pattern for the `M2WSP` WSSPDef.

```
tpaSetActivePattern( "M2WSP" "minWidth")
```

tpaSetAllDefsVisible

```
tpaSetAllDefsVisible(
     g_visible
     )
     => t / nil
```

Description

Sets the visibility of the active width spacing snap pattern definitions (WSSPDefs) to All Visible (AV) or None Visible (NV) for a cellview.

Arguments

g_visible	Boolean variable that sets the visibility of the active WSSPDefs. Valid values: `t` for visible, `nil` for not visible

Value Returned

`t`	The visibility of the active WSSPDefs is set as requested.
`nil`	The visibility of the active WSSPDefs was not set as requested.

Example

Turns the visibility on for the active WSSPDefs.

```
tpaSetAllDefsVisible(t)
```

tpaSetDefVisible

```
tpaSetDefVisible(
     t_defName
     g_visible
     )
     => t / nil
```

Description

Sets the visibility of the active pattern for the given width spacing snap pattern definition (WSSPDef) or snap pattern definition (SPDef) in the Track Pattern table. If the WSSPDef or SPDef is enabled in the Track Pattern table, this also controls the visibility of the active pattern in the canvas.

Arguments

t_defName	Name of the WSSPDef or SPDef.
g_visible	Boolean variable that sets the visibility for the active pattern of the specified WSSPDef or SPDef. Valid values: `t` for visible, `nil` for not visible

Value Returned

`t`	The visibility of the active pattern is set as requested.
`nil`	The visibility of the active pattern was not set because the width spacing pattern or the WSSPDef was not found.

Example

Turns the visibility on for the active pattern of the `M2WSP` WSSPDef.

```
tpaSetDefVisible( "M2WSP" t)
```

tpaSetFilterByName

```
tpaSetFilterByName(
     t_filterName
     )
     => t / nil
```

Description

Sets the related snap pattern (RSP) filter by name or by other filters specified in the *Track Pattern* assistant.

Arguments

t_filterName	Name of the RSP filter.

Value Returned

`t`	The RSP filter is set.
`nil`	The RSP filter is not set.

Examples

Sets the `halfWidthPats` RSP filter in the *Track Pattern* assistant.

```
tpaSetFilterByName("halfWidthPats")
```

tpaSetFilterByNumber

```
tpaSetFilterByNumber(
     x_filterNumber
     )
     => t / nil
```

Description

Sets the Track Pattern Assistant *Filter* field using the list order number for the desired setting.

Arguments

x_filterNumber	Sets the Track Pattern Assistant *Filter* field to the entry in the list order that is represented by this number. If the number is the same as the current *Filter* setting, no action is taken in order to prevent the other filter fields from being overwritten. Valid values: 0 through *n*, where n is the number of entries in the *Filter* list. A value of 0 resets all the Track Pattern Assistant filters and clears the *V* (Visibility) column of the Track Pattern table.

Value Returned

t	Track Pattern Assistant *Filter* field is set to the given value. The Track Pattern table is refreshed using the new filter.
nil	Track Pattern Assistant *Filter* field not set because the number was invalid or was the same as the current filter setting.

Examples

Resets all the Track Pattern Assistant filters and clears the *V* (Visibility) column of the Track Pattern table:

```
tpaSetFilterByNumber(0)
```

Resets all the Track Pattern Assistant filters:

```
tpaSetFilterByNumber(1)
```

Sets the Track Pattern *Filter* to the second entry in the list:

```
tpaSetFilterByNumber(2)
```

tpaSetRegionActivePattern

```
tpaSetRegionActivePattern(
     t_regionName
     t_wsspDefName
     t_wspName
     )
     => t / nil
```

Description

Sets the active width spacing pattern (WSP) of the specified width spacing snap pattern definition (WSSPDef) on a local region.

Arguments

t_regionName	Name of the local region.
t_wsspDefName	Name of the WSSPDef.
t_wspName	Name of the WSP to be set as active.

Value Returned

t	The active pattern was set.
nil	The active pattern was not set.

Example

Sets the active WSP of M2WSP WSSPDef and region FG__0.

```
tpaSetRegionActivePattern("FG__0" "M2WSP" "2XWidth")
=> t
```

tpaSetRegionDefVisible

```
tpaSetRegionDefVisible(
     t_regionName
     t_defName
     g_visible
     )
     => t / nil
```

Description

Sets the visibility of the width spacing snap pattern definition (WSSPDef) or snap pattern definition (SPDef) on a local region.

Arguments

t_regionName	Name of local region.
t_defName	Name of the WSSPDef or SPDef.
g_visible	Boolean variable that sets the visibility for the local region of the specified WSSPDef or SPDef. Valid values: `t` for visible, `nil` for not visible

Value Returned

`t`	The visibility of the WSSPDef or SPDef is set as requested.
`nil`	The visibility of the WSSPDef or SPDef is not set.

Example

Turns the visibility on for the WSSPDef or SPDef of the `M2WSP` WSSPDef and region `FG__0`.

```
tpaSetRegionDefVisible("FG__0" "M2WSP" t)
=> t
```

tpaSetRegionWireType

```
tpaSetRegionWireType(
     t_regionName
     t_wspDefName
     l_wireTypeNames
     )
     => t / nil
```

Description

Sets the active wire types of the specified width spacing snap pattern definition (WSSPDef) on a local region.

Arguments

t_regionName	Name of local region.
t_wspDefName	Name of the WSSPDef.
l_wireTypeNames	List of wire types.

Value Returned

`t`	The wire types were set.
`nil`	The wire types were not set.

Example

Sets the `1X` wire type as the only active wire type for the active WSP of the `M2WSP` WSSPDef and region `FG__0`.

```
tpaSetRegionWireType("FG__0" "M2WSP" '("1X" "3X"))
=> t
```

tpaSetWireType

```
tpaSetWireType(
     t_wspDefName
     l_wireTypeNames
     )
     => t / nil
```

Description

Sets the active wire types for the active width spacing pattern (WSP) of the given width spacing snap pattern definition (WSSPDef).

Arguments

t_wspDefName	Name of the WSSPDef.
l_wireTypeNames	List of wire types.

Value Returned

`t`	Only the specified wire types of the active WSP for the given WSSPDef are and were set active.
`nil`	The wire types were not set because the WSSPDef was not found.

Example

Sets the `1X` wire type as the only active wire type for the active WSP of the `M2WSP` WSSPDef.

```
tpaSetWireType( "M2WSP" list("1X") )
```

tpaToggleDef

```
tpaToggleDef(
     t_defName
     g_enabled
     )
     => t / nil
```

Description

Enables or disables the specified WSSPDef or SPDef.

Arguments

t_defName	Name of the WSSPDef or SPDef.
g_enabled	Boolean variable that enables or disables the specified WSSPDef or SPDef. Valid values: `t` for enable, `nil` for disable

Value Returned

`t`	The specified WSSPDef or SPDef is enabled or disabled, as requested.
`nil`	The specified WSSPDef or SPDef was not found.

Example

Disables the `M2WSP` WSSPDef. This also sets the visibility off for the `M2WSP` active pattern in the Track Pattern table and in the canvas.

```
tpaToggleDef( "M2WSP" nil )
```

tpaToggleLayer

```
tpaToggleLayer(
     t_layerName
     g_visible
     )
     => t / nil
```

Description

Sets the visibility for the active pattern of the enabled WSSPDef or SPDef on the given snapping layer.

Arguments

t_layerName	Name of the layer.
g_visible	Boolean variable that sets the visibility for the active pattern of the enabled WSSPDef or SPDef on the given snapping layer. Valid values: `t` for visible, `nil` for not visible

Value Returned

`t`	The visibility for the active pattern of the enabled WSSPDef or SPDef on the given snapping layer is set, as requested.
`nil`	The specified layer name was not found.

Example

Turns off the `Metal2` layer visibility in the Track Pattern table. This also turns off the active pattern visibility for the enabled WSSPDefs or SPDef on the layer.

```
tpaToggleLayer( "Metal2" nil)
```

tpaToggleRegionLayer

```
tpaToggleRegionLayer(
     t_regionName
     t_layerName
     g_visible
     )
     => t / nil
```

Description

Sets the visibility for the layer of the specified region.

Arguments

t_regionName	Name of local region.
t_layerName	Name of the layer.
g_visible	Boolean variable that sets the visibility for the layer. Valid values: `t` for visible, `nil` for not visible

Value Returned

`t`	The visibility for the layer is set.
`nil`	The visibility for the layer is not set. .

Example

```
tpaToggleLayer( "Metal2" nil) tpaToggleRegionLayer("FG__0" "M1" t) => t
```

Time Tracker Functions

You can measure the time for certain commands in Virtuoso® by defining a scope. For any scope, you can measure the time by using the following SKILL functions:

- ttrGetCurrentScope
- ttrGetScopes
- ttrGetTime
- ttrStartTracking
- ttrStopTracking

You can use the time-tacking data for debugging purposes. For example, you can identify where all the CPU time is spent, the time taken by a particular command, or the operations that are causing slowness in the system.

ttrGetCurrentScope

```
ttrGetCurrentScope(
     )
     => l_scopeName / nil
```

Description

Returns the names of the active scopes. An active scope can be enabled for tracking by using the ttrStartTracking SKILL function.

Arguments

None

Value Returned

l_scopeName	The names of scopes, if they are active.
nil	No scope is being tracked.

Examples

The following code returns `Scope1` as the name of the scope being tracked.

```
ttrStartTracking("Scope1")
ttrGetCurrentScope()
=>("Scope1")
```

The following code returns `nil` because no scope is being tracked.

```
ttrStopTracking("Scope1")
ttrGetCurrentScope()
=>nil
```

Related Topics

Time Tracker Functions

ttrGetScopes

```
ttrGetScopes(
     l_scopeName
     )
     => l_scopeName / nil
```

Description

Returns the list of child scope names for a given scope.

Arguments

l_scopeName	Name of the scope. A scope name can be a string or a list.

Value Returned

l_scopeName	The list of child scope names for a given scope.
nil	The command failed to run.

Examples

Returns the child scope names for the scope `list("Layout" "Move")`.

```
ttrGetScopes(list("Layout" "Move"))
=>(("Layout" "Move" "init")
    ("Layout" "Move" "Done"))
```

Related Topics

Time Tracker Functions

ttrGetTime

```
ttrGetTime(
     l_scopeName
     )
     => l_timeData / nil
```

Description

Returns the accumulated tracking time data for the given scope.

Arguments

l_scopeName	Name of the scope for which the data is retrieved. A scope name can be a string or a list.

Value Returned

l_timeData	Returns the accumulated tracking time data for the given scope name in the following format: (*count CPU_Time list*(*totalElapsedTime minElapsedTime avgElapsedTime maxElapsedTime*)) Where: ■ *count*: Number of times the given scope was activated. A scope can be made active multiple times. ■ *CPU_Time*: Total CPU time spent on the given scope. ■ *totalElapsedTime*: Total time spent on the given scope. ■ *minElapsedTime*: Minimum time spent on the given scope. ■ *avgElapsedTime*: Average time spent on the given scope. ■ *maxElapsedTime*: Maximum time spent on the given scope.
nil	The command failed to run.

Examples

Returns the accumulated time data for the given hierarchical scope named Move.

```
ttrGetTime(list("Layout" "Move"))
```

```
=>(1 -1.680776e+12
    (71736.0 71736.0 71736.0 71736.0)
)
```

Related Topics

Time Tracker Functions

ttrStartTracking

```
ttrStartTracking(
     [ l_scopeName ]
     )
     => l_scopeName / nil
```

Description

Starts tracking the time spent on various operations in the given scope. The tracking continues until it is stopped using ttrStopTracking.

Arguments

l_scopeName	Name of the scope for which tracking needs to be started. If a scope name is not provided, a name is automatically generated. A scope name can be a string or a list.

Value Returned

l_scopeName	Returns the name of the given scope if tracking has started. You must use this scope name if you want to stop the tracking or print the diagnostics.
`nil`	The tracking could not be started.

Examples

The following code starts tracking the time for the scope `Myscope1`. It returns the name of the scope being tracked.

```
ttrStartTracking("Myscope1")
=>("Myscope1")
```

The following code starts the tracking. It automatically generates a scope name and returns the name `Scope0`.

```
ttrStartTracking()
=>("Scope0")
```

Related Topics

Time Tracker Functions

ttrStopTracking

```
ttrStopTracking(
     [ l_scopeName ]
     [ g_printReport ]
     )
     => t / nil
```

Description

Stops tracking the time for the given scope name.

Arguments

l_scopeName Name of the scope that you want to stop tracking. A scope name can be a string or a list.

If the scope name is not provided, tracking is stopped for the currently active scope.

g_printReport

Specifies whether a report needs to be printed for the scope. The default value is `nil`.

Value Returned

`t`	The tracking is stopped. A report is also printed if specified.
`nil`	The tracking could not be stopped.

Examples

The following code stops tracking the currently active scope.

```
ttrStopTracking()
```

The following code stops tracking the scope `Myscope1`.

```
ttrStopTracking("Myscope1")
```

Stops tracking the scope `Myscope2` and prints the report.

```
ttrStopTracking("Myscope2" t)
```

The following code stops tracking the currently active scope and prints the report.

```
ttrStopTracking(nil t)
```

Related Topics

Time Tracker Functions

Application Tier Functions

You can use the following SKILL functions to find out the application tier to which an application belongs and to list the layout applications and view types available in your Virtuoso® environment. For example, you can use `leIsEXLAppOrAbove` to determine whether an application is Virtuoso® Layout Suite EXL (Layout EXL) or a higher tier application.

- leIsEXLAppOrAbove
- leIsLayoutAppOrAbove
- leIsMXLAppOrAbove
- leIsXLAppOrAbove
- leLayoutAppNames
- leLayoutViewTypes

leIsEXLAppOrAbove

```
leIsEXLAppOrAbove(
     t_appName
     )
     => t / nil
```

Description

Specifies whether the given application is Layout EXL or a higher tier application.

Arguments

t_appName	Name of the application.

Value Returned

`t`	The specified application is Layout EXL or a higher tier application.
`nil`	The specified application is not Layout EXL or a higher tier application, or the specified name is invalid.

Example

The following code retrieves the name of the application running in the current window as `VLS-EXL` and returns `t` to indicate that the current application is Layout EXL or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"VLS-EXL"
leIsEXLAppOrAbove(appName)
=>t
```

The following code retrieves the name of the application in the current window as `Virtuoso XL` and returns `nil` to indicate that the current application is not Layout EXL or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"Virtuoso XL"
leIsEXLAppOrAbove(appName)
=>nil
```

Related Topics

Application Tier Functions

leIsLayoutAppOrAbove

```
leIsLayoutAppOrAbove(
     t_appName
     )
     => t / nil
```

Description

Specifies whether the given application is Layout Viewer or a higher tier application.

Arguments

t_appName	Name of the application.

Value Returned

`t`	The specified application is Layout Viewer or a higher tier application.
`nil`	The specified application is not Layout Viewer or a higher tier application, or the specified name is invalid.

Example

The following code retrieves the name of the application running in the current window as `Layout` and returns `t` to indicate that the current application is Layout Viewer or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"Layout"
leIsLayoutAppOrAbove(appName)
=>t
```

The following code returns `nil` as the application name is invalid.

```
leIsLayoutAppOrAbove("VLSapp")
=>nil
```

Related Topics

Application Tier Functions

leIsMXLAppOrAbove

```
leIsMXLAppOrAbove(
     t_appName
     )
     => t / nil
```

Description

Specifies whether the given application is Layout MXL or a higher tier application.

Arguments

t_appName	Name of the application.

Value Returned

`t`	The specified application is Layout MXL or a higher tier application.
`nil`	The specified application is not Layout MXL or a higher tier, or the specified name is invalid.

Example

The following code retrieves the name of the application in the current window as `VLS-MXL` and returns `t` to indicate that the current application is Layout MXL or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"VLS-MXL"
leIsMXLAppOrAbove(appName)
=>t
```

The following code retrieves the application name of the current window as `VLS-EXL` and returns `nil` to indicate that the current application is not Layout MXL or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"VLS-EXL"
leIsMXLAppOrAbove(appName)
=>nil
```

Related Topics

Application Tier Functions

leIsXLAppOrAbove

```
leIsXLAppOrAbove(
     t_appName
     )
     => t / nil
```

Description

Specifies whether the given application is Layout XL or a higher tier application.

Arguments

t_appName	Name of the application.

Value Returned

`t`	The specified application is Layout XL or a higher tier application.
`nil`	The specified application is not Layout XL or a higher tier application, or the specified name is invalid.

Example

The following code retrieves the application name in the current window as `VLS-XL` and returns `t` to indicate that the current application is Layout XL or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"VLS-XL"
leIsXLAppOrAbove(appName)
=>t
```

The following code retrieves the application name in the current window as `Layout` and returns `nil` to indicate that the current application is not Layout XL or a higher tier application.

```
appName = hiGetAppType(hiGetCurrentWindow())
=>"Layout"
leIsXLAppOrAbove(appName)
=>nil
```

Related Topics

Application Tier Functions

leLayoutAppNames

```
leLayoutAppNames(
     [ t_appName ]
     )
     => l_appTiers / nil
```

Description

Returns a list of all the layout application tiers in your Virtuoso® environment. If *appName* is specified, it returns the name of the specified application and the higher tiers.

Arguments

t_appName	Name of the application.

Value Returned

l_appTiers	List of all the layout application tiers: `("Layout" "Virtuoso XL" "VLS-EXL" "VLS-MXL")` If *appName* is specified, the name of the specified application and higher tiers is returned.
`nil`	The specified application name is invalid or the command failed.

Example

For the application `Layout`, the following code returns `Layout` and names of all the application tiers higher than `Layout`.

```
leLayoutAppNames("Layout")
=>("Layout" "Virtuoso XL" "VLS-EXL" "VLS-MXL")
```

For the application `Virtuoso XL,` the following code returns `Virtuoso XL` and names of all the higher tiers.

```
leLayoutAppNames("Virtuoso XL")
=>("Virtuoso XL" "VLS-EXL" "VLS-MXL")
```

Related Topics

Application Tier Functions

leLayoutViewTypes

```
leLayoutViewTypes(
     )
     => l_leViewTypes / nil
```

Description

Returns a list of the layout view types in your Virtuoso® environment.

Arguments

None

Value Returned

l_leViewTypes	List of the layout view types.
nil	The command failed.

Example

The following code returns the list of the layout view types in your Virtuoso® environment.

```
leLayoutViewTypes()
=>("maskLayout" "maskLayoutXL" "maskLayoutEXL" "maskLayoutMXL")
```

Related Topics

Application Tier Functions

2

Connectivity Driven Editing Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso® Layout Suite layout editor.

It includes functions to let you define bindkeys that correspond to menu commands. These functions are designed for use only with bindkeys and are not recommended as a general programmer interface. It also includes functions to manipulate extraction layers and no overlap layers in the `virtuosoDefaultExtractorSetup` constraint group of the technology file.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

lxIsGroupArrayEnabled

lxIsGroupArrayLocked

lxIsGroupArrayRegular

lxHiPRBoundaryInteriorHalo

lxLaunchLayoutEXL

lxLaunchLayoutXL

lxLaunchLayoutMXL

lxMakeDummy

lxMakePrBoundarySelectable

lxMakePrBoundaryUnselectable

lxPermPermutePins

lxProbeRemoveAll

lxRegMasterDiffName

lxRegPostUpdateComponentsAndNets

lxRegPrePosition

lxRegPreUpdateComponentsAndNets

lxRemoveSynchronousCloneFromFamily

lxRunCmdInVXL

lxRunCmdInXL

lxSelectedDeleteNetRouting

lxSelectedExtendChain

lxSelectedExtendExpanded

lxSelectedExtendSchematic

lxSelectedExtendSelection

lxSetGenerateOptions

lxSetGroupArrayLocked

lxSetGroupArrayRegular

lxSetNetPinSpecs

lxSetPreserveFloorplanningOptions

lxSetSchematicDriven

lxSetUpdateOptions

lxShapeSlotting

lxShowCommonIncompleteNetsForSelectedInsts

lxShowHideIncompleteNets

lxToggleLocalAbutment

lxToggleShowIncompleteNets

lxUnfold

lxUnregMasterDiffName

lxUnregPrePosition

lxUnSlotVia

lxUpdateAllPhysBinding

lxUpdateBinding

lxUpdateComponentsAndNets

lxUpdateComponentsAndNetsFinish

lxUpdateComponentsAndNetsStart

lxUpdateGroupArrayParams

lxUpdatePhysBinding

lxUpdatePlacementStatus

lxUpdateSchematicParameters

lxVerifyCloneFamily

nclRunConstraintValidation

nclToggleCAEMode

soiEnableDrcDfm

techGetLxExtractLayers

techGetLxNoOverlapLayers

techIsLxExtractLayer

techIsLxNoOverlapLayer

techSetLxExtractLayer

techSetLxExtractLayers

techSetLxNoOverlapLayer

techSetLxNoOverlapLayers

tpoIsLayoutXL

tpoIsLayoutEXL

abGetAssistant

```
abGetAssistant(
     w_windowID
     )
     => w_windowID
```

Description

Returns the Annotation Browser window associated with the specified window.

Argument

w_windowID	ID of a window.

Value Returned

w_windowID	ID of the Annotation Browser window.

Example

Returns the ID of the current window where Annotation Browser is open.

```
abGetAssistant(hiGetCurrentWindow())
> dwindow:41
```

Related Topics

Annotation Browser Assistant

abGetCurrentIncompleteNetFilter

```
abGetCurrentIncompleteNetFilter(
     w_windowID
     )
     => t_filterState / nil
```

Description

Returns the incomplete net filter currently applied on the specified window. An incomplete net filter is used to control the display of incomplete net markers in the Connectivity tab of the Annotation Browser assistant by showing or hiding a subset of net names.

Arguments

w_windowID	ID of the window.

Value Returned

t_filterState	The incomplete net filter currently set in the Connectivity tab of the Annotation Browser assistant for the specified window.
nil	The specified window ID is invalid or the Annotation Browser assistant is not open in Layout XL.

Example

```
curIncNetFilter = abGetCurrentIncompleteNetFilter(hiGetCurrentWindow())
(nil mode "Show" netNames ("myNet" "net*")
state t
)
curIncNetFilter->mode
"Show"
curIncNetFilter->netNames
("myNet" "net*")
curIncNetFilter->state
t
```

Returns the current incomplete net filter mode, state, and the net names on which the filter is applied.

Related Topics

Incomplete Net Filter

Annotation Browser Assistant

abHiFindMarker

```
abHiFindMarker(
     [ w_windowID ]
     )
     => t / nil
```

Description

In the specified window, selects the row corresponding to a marker in the Annotation Browser when the corresponding marker object is clicked in the layout canvas. Even though the marker object is selected in the layout canvas, this does not modify the selection in the canvas. If the corresponding marker is not displayed in the Annotation Browser due to the applied filtering, nothing is selected in the Annotation Browser. If a window ID is not specified, the current window is used.

Arguments

w_windowID	ID of the window for which the marker clicked in the layout canvas needs to be found in the Annotation Browser.

Value Returned

t	The command was successful.
nil	The command was canceled.

Example

```
abHiFindMarker(hiGetCurrentWindow())
```

Uses the current window to locate in Annotation Browser the marker that corresponds to the marker object clicked in the layout canvas.

Additional Information

The `abHiFindMarker` enter function does not modify the canvas selection. For example, if you select some shapes and instances in the layout canvas and then want to check the corresponding marker, you can invoke the `abHiFindMarker` enter function and click the marker object in the layout canvas. This selects the corresponding marker in the Annotation Browser, but in the layout canvas, it will still be the shapes and instances that are selected, not the marker object.

In some cases, such as when the applied Annotation Browser filters prevent the marker from displaying or when the marker exists outside of the display depth set for the Annotation Browser, the marker clicked in the layout canvas is not selected in the Annotation browser at all. In this case, if a marker was previously selected in the Annotation Browser, it is deselected and no new marker is selected in the Annotation Browser.

Related Topics

Incomplete Net Filter

Annotation Browser Assistant

abSetColorOnSelection

```
abSetColorOnSelection(
     t_packetName
     [ w_window ]
     )
     => t
```

Description

Sets the specified color as the highlight color for all selected rows in the Annotation Browser tree view.

Arguments

t_packetName	Name of the packet color to assign. The packetName must be a packet supported by Annotation Browser, otherwise a warning is displayed.
w_window	The function applies to the Annotation Browser associated with the specified window. The default is the current window.

Value Returned

`t`	The operation was successful.

Examples

Sets `hilite3` as the color used to highlight all selected rows in the Annotation Browser tree view.

```
abSetColorOnSelection("hilite3")
```

Related Topics

Annotation Browser Assistant

abSetIncompleteNetFilter

```
abSetIncompleteNetFilter(
     w_windowID
     t_filterState { Show | Hide }
     l_netNamess
     g_enable
     )
     => t / nil
```

Description

Sets the specified incomplete net filter state for the specified list of nets in the given window. An incomplete net filter is used to control the display of incomplete net markers in the Connectivity tab of the Annotation Browser assistant by showing or hiding a subset of net names.

Arguments

w_windowID

ID of the window.

t_filterState

One of the available incomplete net filters:

- *"Show"* – shows only the markers associated with the listed nets.
- *"Hide"* – hides the markers associated with the listed nets.

The value you specify for the argument should be enclosed in quotation marks.

l_netNames

List of net names for which the `filterState` needs to be set.

Use of wildcards is supported.

g_enable

Specifies if the incomplete net filter should be enabled.

Value Returned

`t`	Sets the specified filter state for the specified set of nets.
`nil`	The specified window ID is invalid or the specified filter state is not a valid incomplete net filter.

Example

Sets a filter to show the incomplete net markers only for nets that match the net name "`myNet`" or any name that starts with "`net`", such as `net1`, `net2`, `net<3>`, and so on.

```
abSetIncompleteNetFilter(hiGetCurrentWindow() "Show" list("net*" "myNet") t)
=> t
```

Related Topics

Incomplete Net Filter

Annotation Browser Assistant

bndAddInstsBindingByName

```
bndAddInstsBindingByName(
     d_schCellViewID
     t_schInstName
     d_layCellViewID
     t_layInstName
     )
     => t / nil
```

Description

Adds a named instance to an existing binding. If no binding exists, a new one is created. The schematic and layout instances must both be leaf instances. You can also use `bndSetInstsBindingByName()` to add the first binding and then add additional bindings to the same source or target instance by using the `bndAddInstBindingByName()` SKILL function.

Arguments

d_schCellViewID

Database ID of the schematic cellview containing the schematic instance.

t_schInstName

Name of the schematic instance to be bound.

d_layCellViewID

Database ID of the layout cellview containing the layout instance.

t_layInstName

Name of the layout instance to be bound.

Value Returned

`t`	The binding was created.
`nil`	The binding was not created.

Example

```
bndAddInstBindingByName(scv "CINV1/N0" lcv "INV1|N0")
```

Creates a binding between schematic instance `INV1/N0` and layout instance `INV1|N0`.

```
bndAddInstsBindingByName(scv "NC<1>/N0" lcv "NC(1)|N0")
```

Creates a binding between individual bit of schematic instances `NC<0:3>`.

```
bndSetInstsBindingByName(scv "P7" lcv "P7.1")
```

Attempts to create an initial binding between the schematic instance "`P7`" and the layout instance "`P7.1`" but issues a warning because the schematic instance has an mfactor=2 set, which requires an additional layout instance to be provided for binding.

WARNING (BND-2008): Binding for instance '`P7`' expects 1 additional layout instances.

```
bndAddInstsBindingByName(scv "P7" lcv "P7.2")
```

Provides an additional layout instance to complete the binding.

Additional Information

On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to `INV1|P0`, but `INV1` instance cannot bind to `INV1|P0`. For transparent instances, such as `INV1`, you can use `INV1/N0` to add a binding to the `nmos` inside.

Related Topics

bndSetInstsBindingByName

leaf instances

bndAddObjectsBinding

```
bndAddObjectsBinding(
     d_schDbID
     d_layDbID

     => t / nil
```

Description

Adds one or more objects to an existing binding. If no binding exists, a new binding is created. The specified objects must all be leaf-level instances. On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to `INV1|P0`, but `INV1` instance cannot bind to `INV1|P0`. For transparent instances, such as `INV1`, you can use `INV1/N0` to add a binding to the `nmos` inside.

Arguments

d_schDbID

Database IDs of the objects to bind, used to specify one or more instances, terminals or shapes at the top level.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

d_layDbID

Database IDs of the layout objects to bind.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

Value Returned

`t`	The binding was created.
`nil`	The binding was not created.

Examples

```
I1=dbCreateInstByMasterName(schCellViewID "testcase" "inv" "symbol" "I1" '(-1.375
1.9375) "R0")
bndAddObjectsBinding(I1 dbFindAnyInstByName(layCellViewID "I0.1|I2(1)"))
t
```

Creates a new schematic instance called `I1` and binds it to layout instance `I0.1|I2(1)`, which is already bound.

```
bndAddObjectsBinding(dbFindAnyInstByName(schCellViewID "I0")
dbFindAnyInstByName(layCV "I0.1|I2(1)"))
*Error* bndAddObjectsBinding: (BND-3026): Cannot add binding to instance 'I0'
because it is not a leaf. Binding can only be set between leaf objects in the
schematic and layout objects.
```

Attempts to create a binding between schematic instance `I0` and layout instance `I0.1|I2(1)`, but fails because `I0` is not a leaf instance.

bndGetBoundObjects

```
bndGetBoundObjects(
     d_dbID
     [ d_layCV ]
     )
     => l_list_of_dbIDs / nil
```

Description

Returns a list of objects bound to a specified object. The object ID of the bound objects can be specified in different formats. See the argument description for more information.

Arguments

d_dbID	Database ID of the object to query. The object ID could be specified as a: ■ **Single database ID:** `dbId` ■ **List of database IDs:** `list(list(d_dbId1 d_dbId2))` ■ **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))` ■ **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))` ■ **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`
d_layCV	Database ID of the layout cellview to be used if the object specified is a schematic object.
`nil`	List of bound objects cannot be returned because the binder is not initialized.

Value Returned

l_list_of_dbIDs

List of lists of database IDs, member instance numbers, rows, and columns of the objects bound to the specified object.

The syntax of the return value is:

```
((dbId membInst row col) (dbId membInst row col)…)
```

where:

- `dbID` is the database ID of the bound object
- `membInst` is the member instance number of the bound schematic vector bit
- `row`/`col` represents the rows and columns of a bound layout mosaic

Example

Returns a list of the objects bound to layout instance `I0.1|I2(1)`.

```
bndGetBoundObjects(dbFindAnyInstByName(layCV "I0.1|I2(1)"))
(((db:0x0a72ea95 0 0 0)
    (db:0x0a72cd9e 1 0 0)
     )
)
```

Returns a list of the database IDs of the terminals bound to schematic instances `a<0:3>`.

```
bndGetBoundObjects(dbFindNetByName(schCV "a<0:3>")~>term layCV)
((db:0x0a738016)
     (db:0x0a738017)
     (db:0x0a738018)
     (db:0x0a738019)
)
bndGetBoundObjects(list(geGetInstHierPath(schWin)) layCV)
(((db:0x0a72ea93 0 0 0)
    (db:0x0a73cd9e 1 0 0)
     )
)
```

Returns a list of instances bound to the schematic bit.

```
vectInst = dbFindAnyInstByName(schCV "I<0:1>")
bit = 1
bndGetBoundObjects(list(list(vectInst bit)))
(((db:0x1e479d9d 0 0 0)))
```

bndGetSiblingBoundObjects

```
bndGetSiblingBoundObjects(
     d_dbID
     [ d_layCV ]
     )
     => ( dbID membInst row col ) / ( list_of_dbIDs )
```

Description

Returns the objects bound in the same many-to-many relationship as the specified object, or the objects bound to the same mfactored, sfactored, folded or `lxCombination` instance as the specified object. However, the specified object is not included in the result. The object ID can be specified in different formats. See the argument description for more information.

Arguments

d_dbID	Database IDs of the object to query. The object ID could be specified as a: ■ **Single database ID:** `dbId` ■ **List of database IDs:** `list(list(d_dbId1 d_dbId2))` ■ **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))` ■ **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))` ■ **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`
d_layCV	Database ID of the layout cellview to be used if the specified object is in the schematic view.

Value Returned

(*dbID membInst row col*)	List of database IDs, member instances, rows, and columns of the objects bound to the specified object.
list_of_dbIDs	List of database IDs for the objects bound to the specified object.

Example

```
bndGetSiblingBoundObjects(dbFindAnyInstByName(layCellViewID "I0.1|I2(1)"))
(((db:0x0a739ba9 0 0 0)))
```

Returns the database ID of a sibling object for layout instance `I0.1|I2(1)`.

```
bndGetSiblingBoundObjects(dbFindNetByName(schCellViewID "a<0:3>")~>term)
((db:0x0a73ed98)
     (db:0x0a73ed99)
     (db:0x0a73ed9a)
     (db:0x0a73ed9b)
)
```

Returns a list of database IDs for the terminals bound in a relationship with schematic instances `a<0:3>`.

bndRemoveInstBindingByName

```
bndRemoveInstBindingByName(
     d_dbCellViewID
     t_objName
     [ d_dbCellViewID ]
     )
     => t / nil
```

Description

Removes the binding of a named instance.

Arguments

d_dbCellViewID

Database ID of the schematic or layout cellview that contains the instance for which binding needs to be removed.

If a schematic cellview ID is passed, the instance name specified is that of the schematic instance. If a layout cellview ID is passed, the instance name specified is that of the layout instance.

t_objName

Name of the instance.

d_dbCellViewID

Additional, optional layout cellview ID in which the instance to be unbound is searched.

Value Returned

`t`	The binding was removed.
`nil`	The binding was not removed.

Example

Removes the binding for a bound layout instance `I145`.

```
bndRemoveInstBindingByName(layCV "I145")
t
```

Layout instance `I145` is unbound.

```
bndRemoveInstBindingByName(layCV "I145")
nil
```

Removes the binding for a bound schematic instance `I12`.

```
bndRemoveInstBindingByName(schCV "I12" layCV)
t
```

bndRegRemoveDeviceGetSurvivingNet

```
bndRegRemoveDeviceGetSurvivingNet(
     S_function
     )
     => t / nil
```

Description

Registers a user-defined SKILL function that takes a list of net IDs and returns the ID of the single net that survives when devices are removed during layout generation.

Arguments

S_function	Name of the user-defined SKILL function that is registered by `bndRegRemoveDeviceGetSurvivingNet`.

Value Returned

`t`	The user-defined SKILL function was registered.
`nil`	The user-defined SKILL function was not registered.

Example

Registers the user-defined SKILL function before launching XL and chooses the surviving net alphabetically.

```
procedure( sortName( i1 i2 )
   alphaNumCmp(i1~>name i2~>name) <= 0
)
procedure(chooseNet(nets)
    let((net)
        net = car(sort(nets 'sortName))
        net
    )
)
bndRegRemoveDeviceGetSurvivingNet('chooseNet)
```

bndRemoveObjectBinding

```
bndRemoveObjectBinding(
     d_dbID
     [ d_layCV ]
     )
     => t / nil
```

Description

Removes the binding of the specified objects. The object ID can be specified in different formats. See the argument description for more information.

Arguments

d_dbID Database ID of the object for which the binding is to be removed.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

d_layCV Database ID of the layout cellview to be used if the specified object is in the schematic.

Value Returned

`t`	The binding was removed.
`nil`	The binding was not removed.

Example

Removes the binding for instance `I1`.

```
I1 = dbGetInstByName(lcv "I1"))
bndRemoveObjectBinding(I1)
```

Removes the binding for instance `I1`, if you have more than one XL session open for the same schematic with a different layout.

```
bndRemoveObjectBinding(I1 lcv)
```

Removes the binding for the `R4<1>` bit of the schematic instance.

```
R4 = dbGetinstByName(scv "R4<0:1>")
bndRemoveObjectBinding(list(list(R4 1)))
```

Removes the binding between `H3<1>` and `NAND/N0`, which is an instance in lower cell in the schematic hierarchy `NAND/N0`.

```
H3 = dbGetAnyInstByName(scv "H3<0:1>")
NANDN0 = dbGetAnyInstByName(scv2 "N0")
bndRemoveObjectBinding(list(list(H3 1) list(NANDN0)))
```

Removes the binding for instances at the hierarchical path returned by `geGetInstHierPath(w_window)`. Note though that instances in the layout window must be selected before they can be used.

```
bndRemoveObjectBinding(geGetInstHierPath(w_layWindow))
```

bndRemoveTermBindingByName

```
bndRemoveTermBindingByName(
     d_cellViewID
     t_name
     )
     => t / nil
```

Description

Removes the binding from a named terminal.

Arguments

d_cellViewID	The database ID of the cellview containing the terminal.
t_name	The name of the terminal for which the binding is to be removed.

Value Returned

t	The binding was removed.
nil	The binding was not removed.

Example

Removes the binding from terminal a<1> in the specified cellview.

```
bndRemoveTermBindingByName(cellViewID "a<1>")
t
```

bndReplaceInstsBindingByName

```
bndReplaceInstsBindingByName(
     d_schCellViewID
     t_schInstName
     d_layCellViewID
     t_layInstName
     )
     => t / nil
```

Description

Sets a binding between specified schematic and layout instances, replacing any existing bindings in the process. The specified instances must all be leaf instances. On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to `INV1|P0`, but `INV1` instance cannot bind to `INV1|P0`. For transparent instances, such as `INV1`, you can use `INV1/N0` to add a binding to the `nmos` inside.

Arguments

d_schCellViewID

Database ID of the top-level schematic cellview containing the schematic instance to be bound.

t_schInstName

Name of the schematic instance to be bound.

d_layCellViewID

Database ID of the layout cellview containing the layout instance to be bound.

t_layInstName

Name of the layout instance to be bound.

Value Returned

`t`	The binding was replaced.
`nil`	The binding was not replaced.

Example

Binds schematic instance `I1` to layout instance `I0.1|I2(1).`

```
bndReplaceInstsBindingByName(schCellViewID "I1" layCellViewID "I0.1|I2(1)")
t
```

Attempts to bind schematic instance `I0` to layout instance `I145,` but fails because `I0` is not a leaf instance.

```
bndReplaceInstsBindingByName(schCellViewID "I0" layCellViewID "I145")
*Error* bndReplaceInstsBindingByName: (BND-3026): Cannot add binding to
instance’I0’ because it is not a leaf. Binding can only be set between leaf objects
in the schematic and layout objects.
```

bndReplaceObjectsBinding

```
bndReplaceObjectsBinding(
     d_schDbID
     d_layDbID
     )
     => t / nil
```

Description

Sets a binding between specified schematic and layout objects, replacing any existing bindings in the process. The specified objects must all be lead objects. On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to `INV1|P0`, but `INV1` instance cannot bind to `INV1|P0`. For transparent instances, such as `INV1`, you can use `INV1/N0` to add a binding to the `nmos` inside.

Arguments

d_schDbID

Database IDs of the schematic objects to bind.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

d_layDbID

Database IDs of the layout objects to bind.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

Value Returned

`t` The binding was created.

`nil` The binding was not created.

Example

Sets a binding between schematic instance `I1` and an already-bound layout instance `I0.1|I2(1)`. The existing binding for `I0.1|I2(1)` is removed.

```
I1=dbCreateInstByMasterName(schCellViewID "testcase" "inv" "symbol" "I1" '(-1.375
1.9375) "R0")
bndReplaceObjectsBinding(I1 dbFindAnyInstByName(layCellViewID "I0.1|I2(1)"))
t
```

Attempts to set a binding between schematic instance `I0` and layout instance `I0.1|I2(1)` but fails because `I0` is not a leaf object.

```
bndReplaceObjectsBinding(dbFindAnyInstByName(schCellViewID "I0")
dbFindAnyInstByName(layCellViewID "I0.1|I2(1)"))
*Error* bndReplaceObjectsBinding: (BND-3026): Cannot add binding to instance 'I0'
because it is not a leaf. Binding can only be set between leaf objects in the
schematic and layout objects.
```

Binds `I1` either to an nmos or a pmos, because `I1` is an inverter.

```
bndReplaceObjectsBinding(i1 list(dbFindAnyInstByName(layCVId "N0")
dbFindAnyInstByName(layCVId "P0"))
```

```
bndReplaceObjectsBinding(list(dbGetInstByName(scv "R0<0:1>") 0)
dbGetInstByName(lcv "R0(0)"))
```

Binds the vector bits on the schematic side.

```
bndReplaceObjectsBinding(list(dbGetInstByName(scv "R0<0:1>") 1)
dbGetInstByName(lcv "R0(1)"))
```

bndReplaceTermsBindingByName

```
bndReplaceTermsBindingByName(
     d_schCellViewID
     t_schTermName
     d_layCellViewID
     t_layTermName
     )
     => t / nil
```

Description

Sets a binding between specified schematic and layout terminals, replacing any existing bindings in the process. The specified terminals must all be leaf objects. On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to `INV1|P0`, but `INV1` instance cannot bind to `INV1|P0`. For transparent instances, such as `INV1`, you can use `INV1/N0` to add a binding to the `nmos` inside.

Arguments

d_schCellViewID	Database ID of the top-level schematic cellview containing the schematic terminal to be bound.
t_schTermName	Name of the schematic terminal to be bound.
d_layCellViewID	Database ID of the layout cellview containing the layout terminal to be bound.
t_layTermName	Name of the layout terminal to be bound.

Value Returned

t	The binding was created.
nil	The binding was not created.

Example

```
bndReplaceTermsBindingByName(schCellViewID "a<0>" layCellViewID "a<2>")
t
```

Binds schematic terminal a<0> in the specified cellview to layout terminal a<2>.

bndSetInstsBindingByName

```
bndSetInstsBindingByName(
     d_schCellViewID
     t_schInstName
     d_layCellViewID
     t_layInstName
     )
     => t / nil
```

Description

Sets a binding between specified schematic and layout instances. Any existing bindings are removed and a new one created. The specified instances must all be leaf instances.

Arguments

d_schCellViewID

Database ID of the top-level schematic cellview containing the schematic instance to be bound.

t_schInstName

Name of the schematic instance to be bound.

d_layCellViewID

Database ID of the layout cellview containing the layout instance to be bound.

t_layInstName

Name of the layout instance to be bound.

Value Returned

`t`	The binding was created.
`nil`	The binding was not created.

Additional Information

On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to

INV1|P0, but INV1 instance cannot bind to INV1|P0. For transparent instances, such as INV1, you can use INV1/N0 to add a binding to the nmos inside.

Example

```
bndSetInstsBindingByName(schCellViewID "I1" layCellViewID "I145")
t
```

Sets a binding between schematic instance I1 and layout instance I145 in the specified cellviews.

```
bndSetInstsBindingByName(schCellViewID "I0" layCellViewID "I145")
*Error* bndSetInstsBindingByName: (BND-3026): Cannot add binding to instance 'I0'
because it is not a leaf. Binding can only be set between leaf objects in the
schematic and layout objects.
```

Attempts to set a binding between schematic instance I0 and layout instance I145, but fails because I0 is not a leaf instance.

```
bndSetInstsBindingByName(schCellViewID "I0/I2<0>" layCellViewID "I145")
*Error* bndSetInstsBindingByName: (BND-3026): Cannot add binding to instance
'I2<0>' because it is not a leaf. Binding can only be set between leaf objects in
the schematic and layout objects.
```

Attempts to set a binding between schematic instance I0/I2<0> and layout instance I145 but fails because I0/I2<0> is not a leaf instance.

Related Topics

leaf instances

bndSetObjectsBinding

```
bndSetObjectsBinding(
     d_schDbID
     d_layDbID
     )
     => t / nil
```

Description

Set a binding between schematic and layout objects. The specified objects must be lead instances and must not be bound already. The object ID can be specified in different formats. See the argument description for more information.

Arguments

d_schDbID

Database IDs of the schematic objects to bind.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

d_layDbID — Database IDs of the layout objects to bind.

The object ID could be specified as a:

- **Single database ID:** `dbId`
- **List of database IDs:** `list(list(d_dbId1 d_dbId2))`
- **Database ID plus an index of a vector bit:** `list(list(d_dbId x_memInst))`
- **Lists of database IDs:** `list(list(d_dbId1) list(dbId2))`
- **Hierarchical path in the format** `list(list(db_id x_memInst x_row x_col))` **returned by** `geGetInstHierPath(w_window)`

Value Returned

`t`	The binding was created.
`nil`	The binding was not created.

Additional Information

On the layout side, a leaf instance is an instance in the top level. On the schematic side, a leaf instance is the level at which the CPH stops such that `INV1/P0` instance can bind to `INV1|P0`, but `INV1` instance cannot bind to `INV1|P0`. For transparent instances, such as `INV1`, you can use `INV1/N0` to add a binding to the `nmos` inside.

Example

```
bndSetObjectsBinding(dbFindAnyInstByName(schCellViewID "I0")
dbFindAnyInstByName(layCellViewID "I0.1|I2(1)"))
*Error* bndSetObjectsBinding: (BND-3026): Cannot add binding to instance 'I0'
because it is not a leaf. Binding can only be set between leaf objects in the
schematic and layout objects.
```

Attempts to bind schematic instance `I0` to layout instance `I0.1|I2(1)`, but fails because `I0` is not a leaf instance.

```
I1=dbCreateInstByMasterName(schCellViewID "testcase" "inv" "symbol" "I1" '(-1.375
1.9375) "R0")
bndSetObjectsBinding(I1 dbFindAnyInstByName(layCellViewID "I0.1|I2(1)"))
*Error* bndSetObjectsBinding: (BND-3023): Cannot add binding to instance
```

```
'|I0.1|I2(1)' because it is already bound. Use 'bndReplaceObjectsBinding' or
'bndReplaceInstsBindingByName' if you want to replace the existing binding.
```

Attempts to bind schematic instance `I1` to layout instance `I0.1|I2(1)`, but fails because `I1` is already bound.

```
bndSetObjectsBinding(list(dbGetInstByName(scv "R0<0:1>") 0) dbGetInstByName(lcv
"Res0"))
```

Binds the schematic vector bits.

```
bndSetObjectsBinding(dbGetInstByName(scv "N0") dbGetInstByName(lcv "N0.1"))
bndAddObjectsBinding(dbGetInstByName(scv "N0") dbGetInstByName(lcv "N0.2"))
```

Binds the schematic instance `N0` to layout instances `N0.1` and `N0.2`.

Related Topics

leaf objects

bndSetTermsBindingByName

```
bndSetTermsBindingByName(
     d_schCellViewID
     t_schTermName
     d_layCellViewID
     t_layTermName
     )
     => t / nil
```

Description

Sets a binding between specified schematic and layout terminals. The specified terminals must not already be bound.

Arguments

d_schCellViewID

Database ID of the top-level schematic cellview containing the schematic terminal to be bound.

t_schTermName

Name of the schematic terminal to be bound.

d_layCellViewID

Database ID of the layout cellview containing the layout terminal to be bound.

t_layTermName

Name of the layout terminal to be bound.

Value Returned

`t`	The binding was created.
`nil`	The binding was not created.

Example

```
bndSetTermsBindingByName(schCellViewID "a<0>" layCellViewID "a<1>")
t
```

Binds terminal a<0> in the specified schematic instance to terminal a<1> in the specified layout instance.

```
bndSetTermsBindingByName(schCellViewID "a<2>" schCellViewID "a<0>")
*Error* bndSetTermsBindingByName: (BND-3024): Cannot add binding to terminal 'a<2>'
because it is already bound. Use 'bndReplaceObjectsBinding' or
'bndReplaceTermsBindingByName' if you want to replace the existing binding.
```

Attempts to bind terminal a<2> in the specified schematic instance to terminal a<0> in the specified layout instance, but fails because a<2> is already bound.

```
bndSetTermsBindingByName(schCellViewID "a<0>" layCellViewID "a<1>")
t
```

Binds terminal a<0> in the specified schematic instance to terminal a<1> in the specified layout instance.

bndUnregRemoveDeviceGetSurvivingNet

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

Description

Unregisters the user-defined SKILL function that is used to get the ID of the surviving net when devices are removed during layout generation.

Arguments

None

Value Returned

`t`	The user-defined SKILL function was unregistered.
`nil`	The command failed.

Example

```
bndUnregRemoveDeviceGetSurvivingNet()
```

dbSetSoftConnectTermConnectToLayer

```
dbSetSoftConnectTermConnectToLayer(
     d_termID
     tx_layer
     )
     => t / nil
```

Description

Sets a connect-to-layer value on a soft-connect terminal that indicates the layer name to which the terminal must connect.

Arguments

d_termID

Terminal ID on which the connect-to-layer attribute needs to be set.

tx_layer

The layer to which the specified soft-connect terminal needs to connect.

The connect-to-layer attribute can be set only on a soft-connect terminal. Setting the value on a non-soft-connect terminal will generate an error. However, if the value set is "", the connect-to-layer attribute is removed.

Value Returned

`t`	The layer name to which the soft-connect terminal must connect is set.
`nil`	The layer name to which the soft-connect terminal must connect is not set.

Example

```
dbSetSoftConnectTermConnectToLayer(term "metal2") => t
```

dbSetSoftConnectTermPinlessLayer

```
dbSetSoftConnectTermPinlessLayer(
     d_termID
     tx_layer
     )
     => t / nil
```

Description

Sets a soft-connect terminal as a pinless layer based on the layer name value you specify. If you specify a text value, the corresponding definition must exist in the current tech database. The pinless layer attribute can be set only on a soft-connect terminal. Setting the value on a non-soft-connect terminal generates an error.

Note: If you specify a layer name value of "", the pinless layer attribute is removed from the soft-connect terminal.

Arguments

d_termID	Terminal ID on which the pinless layer attribute needs to be set.
tx_layer	The layer name value that needs to be made pinless.

Value Returned

`t`	The pinless layer attribute is set.
`nil`	The pinless layer attribute is not set.

Example

```
dbSetSoftConnectTermPinlessLayer(term bulkArea) => t
```

where, `bulkArea` is a derived layer, which can be defined as an intersection of two layers such as poly and diffusion:

```
derivedLayers(bulkArea 10001 (Poly 'and Oxide))
```

Related Topics

Derived Layers

dbSetTermSoftConnect

```
dbSetTermSoftConnect(
     d_termID
     g_value
     )
     => t / nil
```

Description

Sets the soft-connect attribute of an instance terminal to a specific value if the existing value is different from the value you specify.

Note: The terminal on which you set the soft-connect attribute must be a single-bit terminal. Setting the soft-connect attribute on a multi-bit terminal will generate an error.

Arguments

d_termID

Terminal ID on which the soft-connect attribute needs to be set.

g_value

The soft connect value to be set on a layout terminal.

Value Returned

`t`	The soft-connect attribute is set.
`nil`	The soft-connect attribute is not set.

Example

```
dbSetTermSoftConnect(term t)
```

lceAddSimpleStopLayers

```
lceAddSimpleStopLayers(
     libName
     l_LayerNamePairs
     )
     => x_derivedLayers / nil
```

Description

Adds simple derived layers to the `validLayers` section of the technology file that has the `virtuosoDefaultExtractorSetup` constraint group defined. This SKILL function can be useful for adding simple stop layers in memory when the technology file cannot be edited on disk. If a stop layer is added and the second layer has an unknown material, the material is automatically changed to recognition. If you add an invalid stop layer, you must close the design or launch a new Virtuoso session to refresh the memory.

Arguments

libName	Name of the library for which simple derived layers need to be added to the technology file.
l_LayerNamePairs	List of layer-name pairs that need to be added to the `validLayers` section of the technology file for the specified library.

Value Returned

x_derivedLayers	The number of derived stop layers added to the technology file.
`nil`	The command failed.

Example

```
lceAddSimpleStopLayers
("gpdk090" list(list("Oxide" "Poly") list("Metal1" "M1Resdum")))
```

Adds simple derived layers to the `validLayers` section of the `gpdk090` library.

```
procedure(lceAddSimpleStopLayers(libName layerNamePairs)
(let (result(0) lib tech cg constraints validLayers num max layer1 layer2 derived
stop)


;; Use SKILL lceAddSimpleStopLayers
```

```
;;      to libName tech adds in memory derived stop layers for each layerNamePair
;;      and adds the derived stop layers to the virtuosoDefaultExtractorSetup
;;      If a stop layer is added and the second layer has material unknown, it is
;;      changed to material recognition
;;      returns the number of derived stop layers added

lib = ddGetObj(libName)
when( lib
tech = techGetTechFile(lib)
when( tech
cg = cstFindConstraintGroupIn(tech "virtuosoDefaultExtractorSetup")
tech        = cg~>tech
constraints = cg~>objects
when( tech && constraints
validLayers = car(setof(constraint constraints if(constraint~>defName ==
"validLayers" t)))
when( validLayers
num = 1000
max = num * 2
foreach( pair layerNamePairs
layer1 = nth(0 pair)
layer2 = nth(1 pair)
when( stringp(layer1) &&
stringp(layer2) &&
layer1 != layer2 &&
techFindLayer(tech layer1) &&
techFindLayer(tech layer2)
sprintf(derived "lceAddSimpleStopLayers_%s_%s" layer1 layer2)
while( techGetLayerName(tech num) && num < max
num = num + 1
)
when( num < max &&
techCreateDerivedLayer(tech num derived layer1 "not" layer2)
validLayers~>value = cons(derived validLayers~>value)
stop = techFindLayer(tech layer2)
when( stop~>material == "unknown"
stop~>material = "recognition"
)
result = result + 1
)
)
```

```
)
)
)
)
)
result
)
)
```

The procedure below illustrates how the `lceAddSimpleStopLayers` SKILL function can be used to add simple stop layers.

lceClearLogicalConn

```
lceClearLogicalConn(
     d_cellViewID
     )
     => t / nil
```

Description

Removes the logical connectivity on the specified cellview, such as nets that are not attached to terminals, and the nets and `lxStickyNet` property on all routes, shapes, and vias. In addition, the function clears all connectivity on instTerms and removes the markers created by the extractor.

Arguments

d_cellViewID	ID of the cellview for which the logical connectivity needs to be cleared.

Value Returned

`t`	The logical connectivity of the specified cellview was cleared.
`nil`	An error was encountered.

lceDestroyVoidShapes

```
lceDestroyVoidShapes(
     d_cellViewID
     [ ?layerName layerName ]
     )
     => t
```

Description

Destroys void shapes in the given layout cellview.

Arguments

d_cellViewID	ID of the layout cellview for which the void shapes need to be destroyed.
?layerName	Name of the layer for which the void shapes need to be destroyed.

Value Returned

t	The void shapes were destroyed.

Example

```
lceDestroyVoidShapes(cvId)
```

Destroys all void shapes in the specified layout cellview ID.

```
lceDestroyVoidShapes(cvId ?layerName "L01")
```

Destroys all void shapes on the specified layer in the given layout cellview ID.

lceExtract

```
lceExtract(
     d_layCellViewID
     [ ?reportType reportType ]
     [ ?disableAnnotationChecking { t | nil } ]
     )
     => t / nil
```

Description

Runs the Layout XL connectivity extractor on the specified cellview.

Arguments

d_cellViewID	ID of the cellview to be extracted.
?reportType *reportType*	Specifies the type of report printed during extraction, if a report is printed. The default report type is "full". Other supported values for the argument are: "none" and "simple".
?disableAnnotationChecking	Controls whether annotations are checked during extraction. The default is nil.

Value Returned

t	The specified cellview was extracted.
nil	The specified cellview was not extracted.

lceExtractArea

```
lceExtractArea(
     d_layCellViewID
     l_points
     [ ?reportType reportType ]
     )
     => t / nil
```

Description

Extracts connectivity for a specified area of the design.

Arguments

d_layCellViewID	ID of the cellview to be extracted.
l_points	List of coordinates for the area to be extracted.
?reportType *reportType*	Specifies the type of report printed during extraction, if a report is printed. The default report type is "full". Other supported values for the argument are: "none" and "simple".

Value Returned

t	The specified area was extracted.
nil	The specified area was not extracted.

Example

```
lceExtractArea(cvId list(0:0 10:10))
```

Extracts an area of layout cellview cvId with lower left corner at coordinate (0,0) and upper right corner at (10,10).

Related Topics

Extracting Connectivity by Area

lceExtractAreas

```
lceExtractAreas(
     d_layCellViewID
     l_points
     [ ?reportType reportType ]
     )
     => t / nil
```

Description

Extracts connectivity for the specified areas of the design.

Arguments

d_layCellViewID	ID of the cellview to be extracted.
l_points	List of list of coordinates for each area to be extracted.
?reportType *reportType*	Specifies the type of report printed during extraction, if a report is printed. The default report type is "full". Other supported values for the argument are: "none" and "simple".

Value Returned

t	The specified areas were extracted.
nil	The specified areas were not extracted.

Example

```
lceExtractAreas(cvId list(list(0:0 100:200) list(500:0 1000:1000)))
```

For the layout cellview cvId, extracts two areas. The first area extracted is represented using the lower-left corner at coordinate (0,0) and the upper-right corner at (100, 200). The second extracted area is represented using lower-left corner at (500, 0) and the upper-right corner at (1000, 1000).

Related Topics

Extracting Connectivity by Area

lceExtractNet

```
lceExtractNet(
     net
     [ ?reportType reportType ]
     [ ?disableAnnotationChecking {t | nil } ]
     )
     => t / nil
```

Description

Runs the Layout XL connectivity extractor on the specified net.

Arguments

net	Name of the net for which connectivity needs to be extracted.
?reportType *reportType*	Specifies the type of report printed during extraction, if a report is printed. The default report type is "full". Other supported values for the argument are: "none" and "simple".
?disableAnnotationChecking	Controls whether annotations are checked during extraction. The default is nil.

Value Returned

t	The specified net was extracted.
nil	The specified net was not extracted.

Example

```
lceExtractNet(VDD)
```

Runs the Layout XL connectivity extractor on the VDD net.

lceExtractNets

```
lceExtractNets(
     l_nets
     [ ?reportType reportType ]
     [ ?disableAnnotationChecking {t | nil } ]
     )
     => t / nil
```

Description

Runs the Layout XL connectivity extractor on the specified nets.

Arguments

l_nets	List of nets in a cellview that need to be extracted.
?reportType *reportType*	Specifies the type of report printed during extraction, if a report is printed. The default report type is "full". Other supported values for the argument are: "none" and "simple".
?disableAnnotationChecking	Controls whether annotations are checked during extraction. The default is nil.

Value Returned

t	The specified nets were extracted.
nil	The specified nets were not extracted.

Example

```
lceExtractNets(list(VDD net29 net11))
```

Runs the Layout XL connectivity extractor on the nets: VDD net29 net11

lceExtractUnverifiedAreas

```
lceExtractUnverifiedAreas(
     lt_markers
     [ ?reportType reportType ]
     )
     => t / nil
```

Description

Extracts connectivity for the unverified areas that are defined by the bounding box of the specified markers.

Arguments

lt_markers	List of markers that represent the unverified areas.
?reportType *reportType*	Specifies the type of report printed during extraction, if a report is printed. The default report type is "full". Other supported values for the argument are: "none" and "simple".

Value Returned

t	Connectivity extraction could be performed.
nil	Connectivity extraction failed.

Examples

```
lceExtractUnverifiedMarkers(deGetCellView()~>markers)
```

lceGetExtractLayers

```
lceGetExtractLayers(
     d_layCellViewID
     )
     => l_extractLayers / nil
```

Description

Lists the extractable layer-purpose pairs (LPPs) defined for a specified layout cellview. Layout XL can extract through the LPPs that are listed.

The function lists the LPPs specified in the `validLayers` and `validVias` constraints in the `setupConstraintGroup` specified for the layout cellview. If there is no `setupConstraintGroup` set, the function uses the default specified for product `layout` in the environment variable called `setupConstraintGroup`.

When all purposes are allowed for a given layer number, only the layer name is returned. If only a restricted set of purposes is allowed for a given layer number, both layer name and purpose are returned.

Tip

Cadence recommends that you use this function rather than the technology file SKILL function `techGetLxExtractLayers`. This is because `techGetLxExtractLayers` lists the layers specified only in the default `setupConstraintGroup`, namely `virtuosoDefaultExtractorSetup`. `lceGetExtractLayers` widens the search to include the `setupConstraintGroup` set for a specified cellview, which might be different.

Arguments

d_layCellviewID	Database ID of the layout cellview.

Value Returned

l_extractLayers	The list of extractable layer-purpose pairs.
nil	The specified layout cellview does not exist.

Examples

```
lceGetExtractLayers( layCellviewID )
=>  (("poly1" "drawing")
     ("poly1" "pin")
     ("metal1" "drawing")
     ("cont")
    )
```

Returns the extractable layers for a specified layout cellview ID.

Observe that all the purposes of layer `cont` are extractable, only purpose `drawing` is extractable for layer `metal1`, and only purposes `drawing` and `pin` are extractable for layer `poly1`.

lceGetFracturedShapes

```
lceGetFracturedShapes(
     d_dbFigID
     [ g_points ]
     [ g_floatingOnly ]
     [ d_netID ]
     )
     => t / nil
```

Description

Generates a report of the geometry and connectivity of fractured shapes that exist within a figure (shape or prBoundary).

Arguments

d_dbFigID	Database ID of the shape or prBoundary.
g_points	Generates the report only for the fractured shapes overlapping the specified points or a list of points.
g_floatingOnly	Generates the report for floating fractured shapes only. The default is `nil`.
d_netID	Generates the report only for fractured shapes on a specified net.

Value Returned

`t`	The report of fractured shapes.
`nil`	Fractured shapes do not exist.

Examples

```
lceGetFracturedShapes(cv~>prBoundary)
```

Generates a report detailing the geometry and connectivity of fractured shapes within the prBoundary.

```
lceGetFracturedShapes(cv~>prBoundary list('(5 1) '(5 8) '(13 13)))
```

Generates a report detailing the geometry and connectivity of fractured shapes overlapping the specified list of points within the prBoundary.

```
lceGetFracturedShapes(rect 0.5:4)
```

Generates a report detailing the geometry and connectivity of fractured shapes overlapping the specified points within a rectangle.

```
lceGetFracturedShapes(rect nil nil A) ; point=nil, floating only=nil, net=A
```

Generates a report detailing the geometry and connectivity of fractured shapes on `netA` within the specified rectangle.

lceGetFracturedShapesFromNet

```
lceGetFracturedShapesFromNet(
     d_netID
     [ g_substrateOnly ]
     )
     => t / nil
```

Description

Generates a report of fractured shapes corresponding to a specific net.

Arguments

d_netID	ID of the net for which the report on fractured shapes needs to be generated.
g_substrateOnly	A boolean that limits the report to the net IDs corresponding to the substrate. The default is `nil`.

Value Returned

`t`	The report of fractured shapes.
`nil`	Fractured shapes do not exist.

Examples

```
lceGetFracturedShapesFromNet(GND)
```

Generates a report of fractured shapes on the `GND` net.

```
lceGetFracturedShapesFromNet(VDD t)
```

Generates a report of fractured shapes on the substrate that are connected to the `GND` net.

lceGetIncompleteNets

```
lceGetIncompleteNets(
     d_layCellViewID
     )
     => l_netID
```

Description

Returns the list of incomplete net IDs corresponding to the specified cellview.

Arguments

d_layCellviewID	Database ID of the layout cellview.

Value Returned

l_netID	List of incomplete net IDs.
nil	The command failed.

Example

```
lceGetIncompleteNets(cvID)
```

Generates the list of incomplete net IDs for the specified layout cellview.

lceGetIncompleteNetMarkers

```
lceGetIncompleteNetMarkers(
     [ netID | l_netID ]
     )
     => l_incompleteNetMarkerID
```

Description

Generates a list of open markers on the specified incomplete nets. If a single net is specified, the opens on that net are returned. If multiple nets are specified, the opens on each net in the list are returned. If no nets are specified, all the opens in the design are returned.

Arguments

netID | l_netID

ID of the net (or nets) for which the open markers are reported.

- If a single net is specified, the opens on that net are reported.
- If multiple nets are specified, the opens on each net in the list are reported.
- If no nets are specified, all the opens in the design are reported.

Value Returned

l_incompleteNetMarkerID

SKILL list of open marker IDs on the specified nets.

Example

```
a = dbFindNetByName(cv "a")
lceGetIncompleteNetMarkers(a)
```

Returns the open markers on net `a`.

```
b = dbFindNetByName(cv "b")
lceGetIncompleteNetMarkers(list(a b))
```

Returns the open markers on nets `a` and `b`.

```
c = dbFindNetByName(cv "c")
lceGetIncompleteNetMarkers(list(a b c))
```

Returns the open markers on nets `a`, `b`, and `c`.

```
lceGetIncompleteNetMarkers()
```

Returns the open markers in the current edited design.

lceGetNetNamesFromArea

```
lceGetNetNamesFromArea(
     d_layCellViewID
     l_region
     l_layerNames
     [ ?stopLevel x_stopLevel ]
     [ ?maxNumNets n_maxNumNets ]
     [ ?numChasedShapesLimit n_numChasedShapesLimit ]
     )
     => l_netNames
```

Description

Returns a list of the net names corresponding to the effective connectivity of the shapes on the specified layers overlapping a specified area.

Arguments

d_layCellViewID	Database ID of a layout cellview.
l_region	A coordinate representing a point or a list of coordinates representing a rectangle or polygon.
l_layerNames	List of layer names.
?stopLevel *x_stopLevel*	The maximum number of hierarchy levels to search for shapes. The default is `32`.
?maxNumNets *n_maxNumNets*	The maximum number of net names to return. The default is `2`.
?numChasedShapesLimit *n_numChasedShapesLimit*	The maximum number of shapes to chase. The default is `10`.

Value Returned

l_netNames	List of net names.

Examples

```
lceGetNetNamesFromArea(cvId 10:20 "M1")
```

Returns a list of net names corresponding to the effective connectivity of the shapes in the region `10:20` on the layer `"M1"` of the cellview `cvId`.

```
lceGetNetNamesFromArea(cvId list(0:0 10:10) list("M1" "M2"))
```

Returns a list of net names corresponding to the effective connectivity of the shapes in the polygon represented by coordinates `0:0 10:10` on the layers `"M1"` and `"M2"` of the cellview `cvId`.

lceGetMarkerConnectivitySources

```
lceGetMarkerConnectivitySources(
     d_markerId
     ) l_connectivtySources / nil
```

Description

Returns the closest connectivity sources, such as instTerm, pin, and label, corresponding to a marker generated by the connectivity extractor. This marker can be an open or a short.

Arguments

d_markerId	Database ID of a marker.

Value Returned

l_connectivtySources

A list of one or two database IDs corresponding to the connectivity sources.

`nil` If the marker is not recognized or no connectivity source was found

Examples

Returns the closest connectivity sources corresponding to the marker `markerId`.

```
lceGetMarkerConnectivitySource(markerId)
```

lceGetOption

```
lceGetOption(
     name
     [ cvID ]
     )
     => t / nil
```

Description

Returns the value of the specified Connectivity Extractor option for the current or given cellview.

Arguments

name	Name of the connectivity Extractor environment variable for which the value needs to be returned.
cvID	ID of the layout cellview. ■ If specified, the given design must be in Layout XL or a higher tier. ■ If `nil`, take the current edited design in Layout XL or a higher tier.

Value Returned

Value of the specified Connectivity Extractor option.

Example

```
lceGetOption("extractVerifyOpenViolations" cv)
```

Returns the value of the `extractVerifyOpenViolations` environment variable for the specified cellview.

```
lceGetOption("extractStopLevel")
```

Returns the value of the `extractStopLevel` environment variable for the current edited design in Layout XL or a higher tier.

lceGetShortMarkers

```
lceGetShortMarkers(
     [ netID | l_netID ]
     )
     => l_shortMarkerID
```

Description

Prints a list of short markers between the passed nets. If a single net is specified, the shorts between this net and all the other nets are returned. If two nets are specified, the shorts between these two nets are returned. If more nets are specified, the shorts between pairs of nets in the list are returned.

Arguments

netID | l_netID

ID of the net (or nets) for which the short markers are reported.

- If no net is specified, no short marker is reported.
- If a single net is passed, the shorts between this net and all the other nets are reported.
- If two nets are passed, the shorts between these two nets are reported.
- If more nets are passed, the shorts between pairs of nets in the list are reported.

Value Returned

l_shortMarkerID

SKILL list of short marker IDs between the specified nets.

Example

```
a = dbFindNetByName(cv "a")
lceGetShortMarkers(a)
```

Returns the short markers between net `a` and other nets.

```
b = dbFindNetByName(cv "b")
```

```
lceGetShortMarkers(list(a b))
```

Returns the short markers between nets `a` and `b`.

```
c = dbFindNetByName(cv "c")
lceGetShortMarkers(list(a b c))
```

Returns the short markers between net pairs (`a`,`b`), (`a`,`c`), and (`b`,`c`).

```
lceGetShortMarkers()
```

Returns all the short markers in the current edited design.

lceHiExtract

```
lceHiExtract(
     )
     => t
```

Description

Opens the Extract Layout form, which you can use to extract the design either at the top level or hierarchically.

Arguments

None

Value Returned

`t`	The form was opened.

Related Topics

Connectivity Extraction

lceHierExtract

```
lceHierExtract(
     d_cellViewID
     x_searchDepth
     g_extractAll { t | nil }
     g_saveExtractedCellviews { t | nil }
     g_skipTopCellview { t | nil }
     [ g_fromUserList { t | nil } ]
     [ g_fromSelectedInsts { t | nil } ]
     [ g_reportType
     [ g_extractPCells { t | nil } ]
     [ g_disableAnnotationChecking { t | nil } ]
     )
     => t / nil
```

Description

Performs connectivity extraction of a hierarchy of cellviews from bottom to top.

Arguments

d_cellViewID ID of the top cellview to be extracted.

x_searchDepth The maximum depth to search for instance masters to extract in the hierarchy of the top cellview.

g_extractAll Extracts instance masters in the hierarchy.

- When `t`, extracts all instance masters in the hierarchy of the top cellview.
- When `nil`, extracts only instance masters in the hierarchy of the top cellview that are not up-to-date.

g_saveExtractedCellviews

When `t`, saves the extracted cellviews.

g_skipTopCellview

Controls whether the top cellview is extracted.

- When `t`, the top cellview is not extracted.
- When `nil`, the top cellview is extracted after all the sub cells.

g_fromSelectedInsts

Controls which instance masters are extracted.

When `t`, only the instance masters in the hierarchy of the instances selected in the canvas are extracted.

The default is `nil`.

g_fromUserList Extracts only the cellviews that are selected in the Extract Layout form.

The default is `nil`.

g_fromUserList Extracts only the cellviews that are selected in the Extract Layout form.

The default is `nil`.

g_fromSelectedInsts

Controls which instance masters are extracted.

When `t`, only the instance masters in the hierarchy of the instances selected in the canvas are extracted.

The default is `nil`.

g_reportType

Specifies the type of report printed during extraction, if a report is printed.

The default report type is "`full`".

Other supported values for the argument are: "`none`" and "`simple`".

g_extractPCells

Controls whether Pcells are extracted during extraction.

The default is `nil`.

g_disableAnnotationChecking

Controls whether annotations are checked during extraction.

The default is `nil`.

Value Returned

`t`	At least one cellview was extracted.
`nil`	No cellview was extracted.

Example

```
lceHierExtract(cv 32 t nil t) ; extractAll=t, saveExtractedCellviews=nil,
skipTopCellview=t
```

Performs hierarchy extraction for all cellviews, except the top cellview. The extracted cellviews are not saved.

lceIsExtractLayer

```
lceIsExtractLayer(
     d_layCellViewID
     ltx_layer
     )
     => t / nil
```

Description

Checks whether the specified layer or layer-purpose pair (LPP) is extractable.

The function searches for the specified layer or LPP in the `validLayers` and `validVias` constraints in the `setupConstraintGroup` set for a specified layout cellview. If there is no `setupConstraintGroup` set for the specified cellview, the function uses the default specified for product `layout` in the environment variable called `setupConstraintGroup`.

Tip

Cadence recommends that you use this function rather than the technology file SKILL function `techIsLxExtractLayer`. This is because `techIsLxExtractLayer` searches for the specified layer only in the default `setupConstraintGroup`, namely `virtuosoDefaultExtractorSetup`, to determine whether or not the layer can be extracted. `lceIsExtractLayer` widens the search to include the `setupConstraintGroup` set for a specified layout cellview, which might be different.

Arguments

d_techFileID	Database ID of the layout cellview.
ltx_layer	The layer to be checked. Valid Values: A layer name, layer number, or a list containing a layer name and layer purpose.

Value Returned

t	The specified layer or layer-purpose pair can be extracted.
nil	The specified layer or layer-purpose pair cannot be extracted or the specified layout cellview does not exist.

Examples

```
lceIsExtractLayer( layCellViewID "metal1" )
=> t
```

Confirms that layer `metal1` in layout cellview *`layCellViewID`* can be extracted.

```
lceIsExtractLayer( layCellViewID list("metal1" "drawing") )
=> t
```

Confirms that layer-purpose pair `metal1 drawing` in the specified layout cellview can be extracted.

```
lceIsExtractLayer( layCellViewID 45 )
=> t
```

Confirms that layer `45` in layout cellview *`layCellViewID`* can be extracted.

lceIsShapeTrimmed

```
lceIsShapeTrimmed(
     d_shapeId
     )
     => t / nil
```

Description

Checks if the specified shape has been trimmed during connectivity extraction.

Arguments

d_shapeId	Database ID of a shape.

Value Returned

`t`	The shape was trimmed during connectivity extraction.
`nil`	The shape was not trimmed during connectivity extraction or connectivity extraction was not run.

Examples

Checks if the shape `s` has been trimmed during connectivity extraction.

```
lceIsShapeTrimmed(s)
```

lcePrintConstraintGroupWarnings

```
lcePrintConstraintGroupWarnings (
     d_techID
     )
     => t / nil
```

Description

Prints the warnings issued by the connectivity extractor for the specified technology library. The constraint group for which the warnings are issued is the one defined using the setupConstraintGroup environment variable.

Arguments

d_techID

Database ID of the technology library for which the constraint group warnings need to be issued.

Value Returned

`t`	The constraint group could be found.
`nil`	The constraint group could not be found.

Example

```
lcePrintConstraintGroupWarnings(techID)
```

Prints warnings for the specified constraint group for the specified technology library.

This function can be used to print warnings after an ASCII technology file is loaded.

```
procedure(printExtractorWarningsCallback(techId techName)
  lcePrintConstraintGroupWarnings(techId)
)

tcRegPostLoadTrigger('printExtractorWarningsCallback)
```

lcePrintExtractLayers

```
lcePrintExtractLayers(
     d_layCellViewID
     [ ?cgName d_cgName ]
     [ ?verifyWellSubstrateConnections { t | nil } ]
     [ ?useOverlappingLabelsAsConnectivitySources { t | nil } ]
     [ ?enableVirtualConnections { t | nil } ]
     [ ?fileName t_fileName ]
     [ ?appendToFile { t | nil } ]
     )
     => t / nil
```

Description

Generates a report about the extractable layers. See Diagnostics Report (Connectivity form).

Arguments

d_layCellViewID

Layout cellview ID or the technology file ID for which the diagnostics report needs be printed.

?cgName *d_cgName*

Constraint group name.

1. If the cellview is opened in Layout XL and the constraint group name is not specified, the SKILL function uses the constraint group used by the extractor.
2. Otherwise, the constraint group name must be specified.

The default is "".

?verifyWellSubstrateConnections

Indicates if well substrate connections should be considered by the connectivity extractor.

The default is nil.

?useOverlappingLabelsAsConnectivitySources

Indicates if overlapping labels must be used for extraction by the connectivity extractor.

The default is nil.

?enableVirtualConnections

Indicates if virtual connections are supported for extraction.

The default is t.

?fileName *t_fileName*

Indicates if the output should be redirected to the specified file.

The default is "" .

?appendToFile

Indicates if the output should be appended to the specified file.

The default is nil.

Value Returned

`t`	A constraint group was found.
`nil`	No constraint group was found for the specified arguments.

Example

```
lcePrintExtractLayers(cvId ?cgName "cg1" ?fileName "test_output" ?appendToFile t)
```

Generates a report on extractable layers in the specified cellview for the constraint group, `cg1`, and appends the results to the file named, `test_output`.

Related Topics

Connectivity Form

lcePrintExtractVias

```
lcePrintExtractVias(
     d_layCellViewID
     [ ?cgName d_cgName ]
     [ ?verifyWellSubstrateConnections { t | nil } ]
     [ ?useOverlappingLabelsAsConnectivitySources { t | nil } ]
     [ ?enableVirtualConnections { t | nil } ]
     [ ?fileName t_fileName ]
     [ ?appendToFile { t | nil } ]
     )
     => t / nil
```

Description

Generates a report about the extractable vias.

Arguments

d_layCellViewID

Layout cellview ID or the technology file ID for which the diagnostics report needs be printed.

?cgName *d_cgName*

Constraint group name.

1. If the first argument and the cellview is opened in XL and the constraint group name is not specified, the SKILL function uses the constraint group used by the extractor.
2. Otherwise, the constraint group name must be specified.

The default is "".

?verifyWellSubstrateConnections

Indicates if well substrate connections should be considered by the connectivity extractor.

The default is nil.

?useOverlappingLabelsAsConnectivitySources

Indicates if overlapping labels must be used for extraction by the connectivity extractor.

The default is nil.

?enableVirtualConnections

Indicates if virtual connections are supported for extraction.

The default is t.

?fileName *t_fileName*

Indicates if the output should be redirected to the specified file.

The default is "".

?appendToFile

Indicates if the output should be appended to the specified file.

The default is nil.

Value Returned

`t`	A constraint group was found.
`nil`	No constraint group was found for the specified arguments.

Example

```
lcePrintExtractVias(cvId ?cgName "cg1" ?fileName "test_output" ?appendToFile t)
```

Generates a report on extractable vias in the specified cellview for the constraint group, `cg1Name`, and appends the results to the file named, `test_output`.

Related Topics

Connectivity Form

lcePrintTechDiagnostics

```
lcePrintTechDiagnostics(
     d_layCellViewID
     [ ?cgName d_cgName ]
     [ ?verifyWellSubstrateConnections { t | nil } ]
     [ ?useOverlappingLabelsAsConnectivitySources { t | nil } ]
     [ ?enableVirtualConnections { t | nil } ]
     [ ?fileName t_fileName ]
     [ ?appendToFile { t | nil } ]
     )
     => t
```

Description

Generates a report about the connectivity extractor technology diagnostics and indicates if the technology file has been correctly set up for extraction. See *Diagnostics Report*s (Connectivity form).

Arguments

d_layCellViewID

Layout cellview ID or the technology file ID for which the diagnostics report needs be printed.

?cgName *d_cgName*

Constraint group name.

1. If the first argument and the cellview is opened in XL and the constraint group name is not specified, the SKILL function uses the constraint group used by the extractor.
2. Otherwise, the constraint group name must be specified.

The default is "".

?verifyWellSubstrateConnections

Indicates if the diagnostics report must report on well substrate connections.

The default is nil.

?useOverlappingLabelsAsConnectivitySources

Indicates if the diagnostics report must report on overlapping labels.

The default is nil.

?enableVirtualConnections

Indicates if the diagnostics report must report on virtual connections.

The default is t.

?fileName *t_fileName*

Indicates if the output should be redirected to the specified file.

The default is "".

?appendToFile *appendToFile*

Indicates if the output should be appended to the specified file.

The default is nil.

Value Returned

`t`	The technology diagnostics report was generated for the specified layout cellview or technology file.

Example

```
lcePrintTechDiagnostics(techId ?cgName "virtuosoDefaultExtractorSetup"
?verifyWellSusbtrateConnections t ?fileName "output" ?appendToFile t)
```

Generates a diagnostic report for the specified technology file using constraint group `virtuosoDefaultExtractorSetup`. The report is appended to the file named, `output`.

Related Topics

Connectivity Form

lceRegenerateVoidShapes

```
lceRegenerateVoidShapes(
     d_cellViewID
     [ ?layerName layerName ]
     )
     => t
```

Description

Regenerates void shapes in the given layout cellview.

Arguments

d_cellViewID

ID of the layout cellview for which the void shapes need to be regenerated.

?layerName

Name of the layer for which the void shapes need to be regenerated.

Value Returned

t — The void shapes were destroyed or created.

Example

```
lceRegenerateVoidShapes(cvId)
```

Regenerates all void shapes in the specified layout cellview ID.

```
lceRegenerateVoidShapes(cvId ?layerName "L01")
```

Regenerates all void shapes on the specified layer in the given layout cellview ID.

lceSetOption

```
lceSetOption(
     name
     value
     [ cvID ]

     )
     => t / nil
```

Description

Sets the specified Connectivity Extractor option for the current or given cellview.

Arguments

name	Name of the Connectivity Extractor environment variable for which the value needs to be set.
value	The value that needs to be set for the specified Connectivity Extractor option.
cvID	ID of the layout cellview. ■ If specified, the given design must be in Layout XL or a higher tier. ■ If `nil`, take the current edited design in Layout XL or a higher tier.

Value Returned

`t`	The specified Connectivity Extractor option was set.
`nil`	The option was not set.

Example

```
lceSetOption("extractStopLevel" 1)
```

Sets the value of the `extractStopLevel` environment variable for the current edited design to 1.

```
lceSetOption("extractVerifyOpenViolations" nil cv)
```

Sets the value of the `extractVerifyOpenViolations` environment variable for the specfied cellview to `nil`.

lceShortLocatorChaseAndSave

```
lceShortLocatorChaseAndSave(
     d_winID
     g_shapes
     g_hierStart
     t_viewName
     )
     => t / nil
```

Description

Chases islands from the specified start shapes and saves them along with the shortest paths between their points of stickiness.

Arguments

d_winID	ID of the layout window in which you want the function to operate.
g_shapes	List of start shapes that are either in the current edited cellview or in the hierarchy that define the start from where the points of stickiness are chased and saved along their shortest path.
g_hierStart	Hierarchical start level at which the points of stickiness are chased. Any start shapes that are above the specified hierarchical level are ignored during the chase.
t_viewName	A string that represents the saved view name. **Note:** The library and cell name are the same as the current edited cellview.

Value Returned

`t`	At least one shape was saved in the specified view.
`nil`	No shape could be saved.

Examples

Example 1

```
winId = hiGetCurrentWindow()
```

```
geSelectPoint(winId 2:2.5)
r1 = css()
lceShortLocatorChaseAndSave(winId list(r1) 0 "shortLocator")
```

Chases islands from the specified top-level shape and saves the chased view as `shortlocator`.

Example 2

```
cv = geGetEditCellView(winId)
i0 = dbFindAnyInstByName(cv "i0")
r2 = nth(2 i0~>master~>shapes)
lceShortLocatorChaseAndSave(winLay list(r1 list(r2 list(i0))) 0 "shortLocator")
```

Chases islands from the specified shapes, `r1` at top level and `r2` in instance `i0`, and saves the chased view as `shortlocator`.

Example 3

```
m0 = dbFindAnyInstByName(i0~>master "M0")
r3 = nth(7 m0~>master~>shapes)
lceShortLocatorChaseAndSave(winLay list(r1 list(r2 list(i0)) list(r3 list(i0
list(m0 0 1)))) 0 "shortLocator")
```

Chases islands from the specified shapes, `r1` at top level, `r2` in instance `i0`, r3 in the tile 0:1 of mosaic `i0.m0`, and saves the chased view as `shortlocator`.

lceShortLocatorRaiseForm

```
lceShortLocatorRaiseForm(
     )
     => t
```

Description

Opens the Short Locator form, which you can use to locate the shapes in a layout that are causing shorts.

Arguments

None

Value Returned

`t`	The form was opened.

lntAddTrace

```
lntAddTrace(
     w_window
     { d_figId | t_netName }
     [ ?mode t_mode ]
     [ ?hierPath l_hierPath ]
     [ ?startLevel x_startLevel ]
     [ ?stopLevel x_stopLevel ]
     [ ?userArea l_userArea ]
     [ ?hilightIncrementally g_hilightIncrementally ]
     [ ?retainColornStateInfo g_retainColornStateInfo ]
     [ ?hilightColor t_hilightColor ]
     [ ?maxCutShapes t_maxCutShapes ]
     [ ?constraintGroup t_constraintGroup ]
     [ ?stackLayers l_stackLayers ]
     )
     => x_traceId / nil
```

Description

Creates a trace starting from the specified figure or net name in the specified window.

Arguments

w_window	ID of the window in which the trace needs to be created.
t_mode	Specifies how the figures are looked for during tracing. Valid values: `"physical"`, `"logical"` Physical mode strictly uses layer overlaps to determine connectivity. Logical mode uses net assignments.
d_figId	ID of the specified figure from which the trace will be created.
t_netName	Name of the specified net on which the trace will be created.
`?hierPath` *l_hierPath*	Path in the hierarchy at which the trace needs to be created.
`?startLevel` *x_startLevel*	The hierarchy level at which the trace will be started.
`?stopLevel` *x_stopLevel*	The hierarchy level at which the trace will be stopped.
`?userArea` *l_userArea*	Allows defining a region (polygon) that supports restricted tracing. Only those traces are created that have their starting figure inside the defined region.
`?hilightIncrementally` *g_hilightIncrementally*	Controls whether the traced nets for connected shapes are highlighted in the increments of 10K shapes.
`?retainColornStateInfo` *g_retainColornStateInfo*	Retains the color state and lock status information for the figure from which the trace is created.
`?hilightColor` *t_hilightColor*	Specifies the color to be used for creating the trace. The default is `"Cycle"`, which selects the next new color from the associated palette.

`?maxCutShapes` *`t_maxCutShapes`*

Controls the maximum number of cut shapes that are traced per overlap.

Choose from:

- *Low* – 100 cut shapes
- *Medium* – 1000 cut shapes
- *High* – 10000
- *Full* – all cut shapes

The default is *Medium.*

`?constraintGroup` *`t_constraintGroup`*

Specifies the constraint group that defines the via layer, connected layer, and stop layers to be used for creating traces.

Choose from `virtuosoDefaultExtractorSetUp`—the default constraint group used by the Layout XL connectivity extractor—or define a custom constraint group.

The default is `virtuosoDefaultExtractorSetUp`.

`?stackLayers` *`l_stackLayers`*

Specifies the start and end layers to narrow down to a smaller set of extractable layers to be used for tracing. The same layer can be used to define the start and end layer for tracing. This argument is useful when creating a big trace, such as for a power or ground net.

Value Returned

`x_traceId`	ID of the created trace.
`nil`	The trace could not be created.

Examples

```
win = hiGetCurrentWindow()
fig = css()
```

Uses the database ID of any valid figure — `rect`, `path`, `pathSeg`, `circle`, `donut`, `ellipse` or `via` for creating the trace.

```
fig2 = figId
```

//Database figure ID of the figure at depth 2.

```
netName = "n1"
```

// Name of the net from which the trace will be created.

```
hp = ((db:0x3233c71c 0 0 0) (db:ox3cbc819a 0 0 0))
```

Uses the SKILL function `geGetInstHierPath(win)` to derive the hierarchical path of the instance.

```
lntAddTrace(win figId) => 1
```

Searches the edit cellview for the figure with the specified ID.

```
lntAddTrace(win netName)  => 2
```

Searches the edit cellview for the net with the specified name.

```
lntAddTrace(win fig2 ?hierPath hp) => 10
```

Uses the SKILL function from top to create a trace starting with the figure at hierarchy depth 2.

```
lntAddTrace(win n1 ?hierPath hp) => 10
```

Uses the SKILL function from top to create a trace starting with the net n1 from the cellview of the instance at hierarchy depth 2.

```
lntAddTrace(win fig ?stopLevel 20)  => 20
```

Creates a trace starting from the figure at depth 0 and goes up to depth 20.

```
lntAddTrace(win fig ?retainColornStateInfo t) =>21
```

Retains the color and lock status of the chased figure.

```
lntAddTrace(win fig ?stopLevel 20 ?constraintGroup "myconstaintGroup) => 67
```

Uses the custom constraint group, "`myconstaintGroup`", to derive the layer rules to use for tracing.

```
lntAddTrace(win fig2) => nil
```

Creates no trace because the figure to use is present at depth 2, and the hierarchical path to the figure is not supplied.

```
lntAddTrace(hiGetCurrentWindow() figId ?stackLayers list("Metal1" "Metal5"))
```

Traces shapes only on `Metal1` and `Metal5` layers.

```
lntAddTrace(hiGetCurrentWindow() figId ?stackLayers list("Metal1"))
```

Traces shapes only on `Metal1` layer.

Related Topics

Tracing Nets

lntClearNeighbors

```
lntClearNeighbors(
     w_window
     )
     => t /nil
```

Description

Deletes all computed neighbors in the specified window.

Arguments

w_window	ID of the window to be cleared.

Value Returned

`t`	Computed neighbors in the specified window were deleted.
`nil`	Specified window has no computed neighbors.

Example

```
window = hiGetCurrentWindow()
lntComputeNeighbors(window 0.04)
lntClearNeighbors(window)
t
```

Deletes the computed neighbors in the current window.

Related Topics

Tracing Nets

lntComputeNeighbors

```
lntComputeNeighbors(
     w_window
     n_distance
     [ s_layers ]
     [ s_netNameMethod ]
     )
     => t
```

Description

Finds neighboring shapes for the existing traces in the specified window, within the given distance, on the same layer or any extractable layer and creates highlights for the overlapped portions.

Arguments

w_window	ID of the window in which neighbors are to be computed for existing traces.
n_distance	Distance within which overlapping shapes that are not already part of the trace will be highlighted.
s_layers	Layer on which the neighboring shapes of the traced shapes are highlighted. Valid values are `'same` or `'any`. The default is `'same`.
s_netNameMethod	Method for computing the net name for neighboring shapes of the traced shapes. Valid values are `'logical` or `'logicalandphysical`. The default is `'logical`. (Layout EXL Only)

Value Returned

`t`	Neighbors were successfully computed and highlighted.
`nil`	The operation was unsuccessful.

Example

```
window = hiGetCurrentWindow()
;;Create some traces using Net Tracer
lntComputeNeighbors(window 0.04 ?layers 'any ?netNameMethod 'logicalandphysical)
lntClearNeighbors(window)
```

Related Topics

Tracing Nets

lntContSteps

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

Description

Generates the complete trace from the current step to the last when the Net Tracer is invoked in Step trace mode.

Arguments

None

Value Returned

`t`	The continuous Step trace was generated.
`nil`	The continuous Step trace was not generated.

Example

```
lntGetCurrentStep() ==> 1
```

Returns the current step number.

```
lntNextStep()
lntGetCurrentStep() ==> 2
```

Generates the trace for the next step and returns the latest current step number.

```
lntNextStep()
lntGetCurrentStep() ==> 3
```

Generates the trace for the next step and returns the latest current step number.

```
lntContSteps()
lntGetCurrentStep() ==> 10
```

Generates the continuous trace covering all the required steps and returns the latest current step number.

Related Topics

Tracing Nets

lntGetAllTraces

```
lntGetAllTraces(
     d_cellviewID
     )
     => l_traceIDs / nil
```

Description

Lists all the traces in the specified cellview.

Arguments

d_cellviewID	Database ID of the cellview to be queried.

Value Returned

l_traceIDs	List of trace IDs; for example: `(8 7 6 5)`
`nil`	Cellview contains no traces or the operation was unsuccessful.

Example

```
myTraces = nil
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    myTraces = lntGetAllTraces(cellView)
    when(myTraces
      foreach(trace myTraces lntGetTraceInfo(trace))
    )
  )
)
```

Lists the IDs of all the traces in current cellview.

Related Topics

Tracing Nets

lntGetCurrentStep

```
lntGetCurrentStep(
     )
     => x_val
```

Description

Returns the current step value when the Net Tracer is invoked in Step trace mode.

Arguments

None

Value Returned

x_val	Value of the current step number for a Step trace.

Example

```
lntGetCurrentStep() ==> 1
```

Returns the value of the current step number for the trace.

```
lntNextStep()
lntGetCurrentStep() ==> 2
```

Increments the step count by one and returns the latest current step number for the trace.

```
lntContSteps()
lntGetCurrentStep() ==> 10
```

Generates a continuous Step trace up to the last step and returns the latest current value of the step.

Related Topics

Tracing Nets

lntGetSavedViewNames

```
lntGetSavedViewNames(
     t_libName
     t_cellName
     )
     => l_viewNames / nil
```

Description

Returns the list of views saved in the specified library and cell.

Arguments

t_libName	Name of the library that contains the saved views.
t_cellName	Name of the cell that contains the saved views.

Value Returned

l_viewNames	List of views saved in the specified library and cell.
nil	No saved view exists.

Example

The following example returns the list of saved views in the `test1` cell of the `gpdk090` library.

```
lntGetSavedViewNames("gpdk090" "test1")
=> ("netTracer_4" "netTracer_3" "netTracer_2" "netTracer_1")
```

The following example returns `nil` as there are no saved views in the `test2` cell of the `gpdk090` library.

```
lntGetSavedViewNames("gpdk090" "test2")
=> nil
```

Related Topics

lntSaveTraces

lntGetTailVal

```
lntGetTailVal(
     )
     => x_val
```

Description

Returns the current value of the tail of a trace when the Net Tracer is invoked in Step trace mode.

Arguments

None

Value Returned

x_val	Current value of the tail for a Step trace. If a tail value is returned, the value is > 0.

Example

```
lntGetTailVal() ==> 1
```

Returns the current value of the tail for the selected Step trace.

```
lntSetTailVal(5)
lntGetTailVal() ==> 5
```

Sets the tail value of the Step trace to 5 and then returns the newly revised current tail value.

Related Topics

Tracing Nets

lntGetTraceInfo

```
lntGetTraceInfo(
     x_traceID
     )
     => l_paramList / nil
```

Description

Returns information for a specified trace. The returned information contains the trace container, its name, color, and visibility status, and the number of chased shapes.

Arguments

x_traceID	ID of the trace to be queried.

Value Returned

l_paramList	List of parameters for the specified trace; for example: `(db:0x31670f9a "Trace1" "white" t 3)`
`nil`	The operation was unsuccessful.

Example

```
myTraces = nil
when(window = hiGetCurrentWindow()
  when(cellView = geGetWindowCellView(window)
    myTraces =  lntGetAllTraces(cellView)
    when(myTraces
      foreach(mapcar trace myTraces lntGetTraceInfo(trace))
    )
  )
)
```

Returns information about all the traces in the current cellview.

Related Topics

Tracing Nets

lntHideNeighbors

```
lntHideNeighbors(
     w_window
     )
     => t / nil
```

Description

Hides displayed neighbors for the specified window. The computed neighbors are retained and can be displayed again using the `lntShowNeighbors` command. Use `lntClearNeighbors` to delete all neighbors.

Arguments

w_window	ID of the window in which computed neighbors are to be hidden.

Value Returned

`t`	Computed neighbors in the specified window are hidden.
`nil`	Specified window contains no computed neighbors.

Example

```
window = hiGetCurrentWindow()
lntComputeNeighbors(window 0.04)
lntHideNeighbors(window)
lntShowNeighbors(window)
```

Computes and highlights neighbors for the traces in the current window, then hides them, then shows them again.

Related Topics

Tracing Nets

Finding Neighboring Shapes of a Traced Shape

lntHiEditTrace

```
lntHiEditTrace(
     [ w_window ]
     )
     => t / nil
```

Description

Invokes the Net Tracer *Edit In Place* command that lets you edit in place the instances that contain the traced objects.

Arguments

w_window	ID of the window in which the Edit In Place command needs to be run. When a window ID is not specified, the Net Tracer operates on the current layout window.

Value Returned

`t`	The Net Tracer Edit In Place command was invoked.
`nil`	The Net Tracer Edit In Place command was not invoked.

Example

```
lntHiEditTrace()
```

Invokes the Net Tracer Edit In Place command for the layout cellview displayed in the current window.

Related Topics

Tracing Nets

Trace Manager

lntHiNetTracer

```
lntHiNetTracer(
     [ w_window ]
     )
     => t / nil
```

Description

Invokes the Net Tracer command, which you can use to add, update, or delete traces in a layout window.

Arguments

w_window	ID of the window on which the Net Tracer is to operate. When not specified, the Net Tracer operates on the current layout window.

Value Returned

`t`	Net Tracer was invoked.
`nil`	Net Tracer could not be invoked. Check that the current window is a layout window.

Example

```
lntHiNetTracer()
```

Invokes the Net Tracer for the layout cellview displayed in the current window.

Related Topics

Tracing Nets

Net Tracer Options

lntIsNeighborsVisible

```
lntIsNeighborsVisible(
     w_window
     )
     => t / nil
```

Description

Checks whether Net Tracer neighbors are visible in the specified window.

Arguments

w_window	ID of the window where visibility of neighbors is to be checked.

Value Returned

t	The window contains neighbors that are visible.
nil	The window contains neighbors that are set as invisible or does not contain any neighbors.

Example

```
window = hiGetCurrentWindow()
;; Create some traces using Net Tracer
lntComputeNeighbors(window 0.04)
isVisible = lntIsNeighborsVisible(window)
```

Computes trace neighbors in the current window and then checks that they are visible.

Related Topics

Tracing Nets

Finding Neighboring Shapes of a Traced Shape

lntIsStepTrace

```
lntIsStepTrace(
     x_traceID
     )
     => t / nil
```

Description

Checks whether the specified trace was created using Step mode.

Arguments

x_traceID	ID of the trace that is checked.

Value Returned

`t`	The trace was created using Step mode.
`nil`	The trace was not created using Step mode.

Example

```
win = hiGetCurrentWindow()
fig = css()
trace1 = lntAddTrace(win fig)
lntIsStepTrace(trace1) => nil
```

The SKILL function checks the specified trace and determines that the trace was created in normal mode.

In the Net Tracer toolbar, select *Add/Remove step trace* from the mode selection drop-down menu and select the shape to be traced.

```
trace2 = 2 created
lntIsStepTrace(trace2) => t
```

The SKILL function checks the specified trace and determines that the trace was created in step mode.

Related Topics

Tracing Nets

Finding Neighboring Shapes of a Traced Shape

lntNeighbors

```
lntNeighbors(
     w_window
     )
     => t / nil
```

Description

Displays the Net Tracer Neighbors form, which you can use to compute all neighboring shapes within a given distance for all the traces in the window.

Arguments

w_window	ID of the window containing existing traces.

Value Returned

`t`	The Net Tracer Neighbors form is displayed.
`nil`	The Net Tracer Neighbors form could not be opened.

Example

```
lntNeighbors(hiGetCurrentWindow())
```

Opens the Net Tracer Neighbors form.

Related Topics

Tracing Nets

Finding Neighboring Shapes of a Traced Shape

Net Tracer Neighbors Form

lntNextStep

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

Description

Increments the current step value by one when the Net Tracer is invoked in Step trace mode.

Arguments

None

Value Returned

`t`	The trace is incremented by a step.
`nil`	The trace is not incremented by a step.

Example

```
lntGetCurrentStep() ==> 1
lntNextStep()
lntGetCurrentStep() ==> 2
```

Returns the current step value of 1, increments the step value by 1, and then returns the revised current step value of 2.

Related Topics

Tracing Nets

lntNextToSetStep

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

Description

Forwards tracing of all Step mode traces to the newly specified step value. For information about how the SKILL function behaves differently for each trace, depending on the maximum step value for the trace, see Additional Information.

Arguments

None

Value Returned

`t`	The Step trace is forwarded to the newly specified step value.
`nil`	The Step trace is not forwarded to the newly specified step value.

Example

```
lntGetCurrentStep() ==> 1
lntSetCurrentStep() ==> 5
lntGetCurrentStep() ==> 5
lntNextToSetStep()
```

Returns the current step value of 1, sets the new step value to 5, gets the newly specfied step value of 5, and then forwards the trace to step 5.

Additional Information

When the Net Tracer is invoked in Step trace mode, the number of steps that two traces may require to finish the tracing, also called the maximum number steps for a trace, can vary. For example, a short trace may require 10 steps to finish chasing but a long trace may require 20 steps to full shape-chase or trace.

Let us assume that a design has three traces to chase in Step trace mode.

- Trace 1 requires a maximum of 21 steps to reach the finish.
- Trace 2 requires a maximum of 4 steps to reach the finish.
- Trace 3 requires a maximum of 15 steps to reach the finish.
- The current step value, which applies to all the traces, is 2.

Currently, all the three traces are at step 2. Now, if the current step value is changed to 10 and the SKILL function `lntNextToSetStep()` is called, chasing for all the three traces progresses forward.

- Traces 1 and 3 progress forward up to step 10 because the maximum step value for both the traces is greater than 10.
- Trace 2 progresses forward up to step 4 because the maximum step value for this trace is 4.

Now, if the current step value is changed to 6 and the SKILL function `lntNextToSetStep()` is called, Trace 1 and Trace 3 move backward (from step10) to step 6. But, there is no change to Trace 2.

Related Topics

Tracing Nets

lntPrevStep

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

Description

Decrements the current step value by one when the Net Tracer is invoked in Step trace mode.

Arguments

None

Value Returned

`t`	The trace is decremented by a step.
`nil`	The trace is not decremented by a step.

Example

```
lntGetCurrentStep() ==> 1
lntNextStep()
lntGetCurrentStep() ==> 2
```

Returns the current step value of 1, increments the step value by 1, and then returns the revised current step value of 2.

```
lntNextStep()
lntGetCurrentStep() ==> 3
lntNextStep()
lntGetCurrentStep() ==> 4
```

Increments the step value twice and returns the latest current step value of 4.

```
lntPrevStep()
lntGetCurrentStep() ==> 3
```

Decrements the step value and returns the latest current step value of 3.

Related Topics

Tracing Nets

lntRemoveAll

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

Description

Removes all traces created by the Net Tracer command from the current layout window.

Arguments

None

Value Returned

`t`	All traces were removed.
`nil`	Traces could not be removed or the current window is not a layout window.

Example

```
lntRemoveAll()
```

Related Topics

Tracing Nets

Removing All Traces

lntRemoveAllTraces

```
lntRemoveAllTraces(
     [ w_window ]
     )
     => t / nil
```

Description

Removes all traces from the specified window.

Arguments

w_window	ID of the window in which traces are to be removed. When not specified, traces are removed from the current layout window.

Value Returned

`t`	All traces in the specified window were successfully removed.
`nil`	The operation was unsuccessful.

Example

```
lntRemoveAllTraces(hiGetCurrentWindow())
```

Removes all traces from the cellview shown in the current window.

Related Topics

Tracing Nets

Removing All Traces

lntRemoveTrace

```
lntRemoveTrace(
     x_traceID
     [ w_window ]
     )
     => t / nil
```

Description

Removes the specified trace from the specified window.

Arguments

x_traceID	ID of the trace to be removed.
w_window	ID of the window from which the trace is to be removed. When not specified, the trace (if it exists) is removed from the current window.

Value Returned

`t`	Specified trace was removed.
`nil`	Specified trace could not be removed.

Example

```
lntRemoveTrace(1 window(2))
```

Removes the trace with ID `1` from window 2.

Related Topics

Tracing Nets

lntSaveTraces

```
lntSaveTraces(
     w_windowID
     l_traceID
     libName
     cellName
     viewName
     [ g_saveNeighbors ]
     [ g_appendTraces ]
     [ g_replaceSameNameTrace ]
     [ ?oneViewPerTrace g_oneViewPerTrace ]
     )
     => t / nil
```

Description

Saves the shapes of the specified traces that are displayed in the library cellview open in the specified window.

Arguments

w_windowID	ID of the window in which the source cellview is open. The traces to be saved are open in this window.
l_traceID	IDs of the traces belonging to the specified cellview that need to be saved. Trace IDs belonging to windows other than the specified window are ignored, and a warning message is issued.
libName	Name of the target library to which the shapes of the traces will be saved. The source and the target technology set up must be compatible for the traces to be saved.
cellName	Name of the target cell to which the shapes of the traces will be saved. The source and the target technology set up must be compatible for the traces to be saved.

viewName	Name of the target view to which the shapes of the traces will be saved. The source and the target technology set up must be compatible for the traces to be saved.
g_saveNeighbors	Saves the shapes of neighboring traces also. The default is `nil`.
g_appendTraces	Opens the target cellview in `append` mode and adds the shapes of all the specified trace IDs to the cellview. The default is `nil`.
g_replaceSameNameTrace	Deletes those trace shapes from the target cellview that have the same name and adds shapes corresponding to the specified traces. For this argument to take effect, the `appendTraces` argument must be set to `t`. The default is `nil`.
`?oneViewPerTrace` *g_oneViewPerTrace*	Saves each visible trace in a design as a separate view. The default is `nil`.

Value Returned

`t`	The specified traces were saved.
`nil`	No traces were saved.

Examples

```
win = hiGetCurrentWindow()
    cv = geGetEditCellView()
    traceIds1 = list(1 2)
    traceIds2 = list(3 4 5)
    traceIds3 = list(2 1 5 6)
    lntSaveTraces(w traceIds1 cv->libName  cv->cellName "trace1")
    lntSaveTraces(w traceIds2 cv->libName  cv->cellName "trace2" t)
    lntSaveTraces(w traceIds3 cv->libName  cv->cellName "trace3" nil t)
    lntSaveTraces(w traceIds3 cv->libName  cv->cellName "trace1" nil t)
```

```
    lntSaveTraces(w traceIds3 cv->libName  cv->cellName "trace1" nil t  t)
    lntSaveTraces(w traceIds3 cv->libName  cv->cellName "allTraces" nil t t
?oneViewPerTrace t)
```

Related Topics

Tracing Nets

Save Traced Nets View Form

lntSetCurrentStep

```
lntSetCurrentStep(
     x_val
     )
     => t / nil
```

Description

Sets the current step value for a trace when the Net Tracer is invoked in Step trace mode.

Arguments

x_val	New current value to be set for the Step trace.

Value Returned

`t`	The Step trace was set to the new current value.
`nil`	The Step trace was not set to the new current value.

Example

```
lntSetCurrentStep(5) ==> t
lntGetCurrentStep() ==> 5
```

Sets the current step value to 5 and returns the value when `lntGetCurrentStep()` is used.

Related Topics

lntGetCurrentStep

Tracing Nets

lntSetTailVal

```
lntSetTailVal(
     x_val
     )
     => t / nil
```

Description

Sets the current value of the tail for all traces when the Net Tracer is invoked in Step trace mode.

Arguments

`x_val`	Tail value to be set for the Step traces. The specified tail value must be a positive integer or 0.

Value Returned

`t`	The specified tail value was set.
`nil`	The specified tail value was not set.

Example

```
lntGetTailVal() ==> 1
```

Returns the current value of the tail for the selected Step trace.

```
lntSetTailVal(5)
    lntGetTailVal() ==> 5
```

Sets the tail value of the Step trace to 5 and then returns the newly revised current tail value.

Related Topics

Tracing Nets

lntSetTraceColor

```
lntSetTraceColor(
     x_traceID
     t_color
     [ w_window ]
     )
     => t / nil
```

Description

Changes the color of the specified trace to the specified color in the given window.

Arguments

x_traceID	ID of the trace for which the color is to be changed.
t_color	Color to be applied to the specified trace. Supported colors: `blue`, `cyan`, `lime`, `magenta`, `orange`, `pink`, `purple`, `red`, `yellow`
w_window	ID of the window containing the specified trace. When not specified, the trace (if it exists) is updated in the current window.

Value Returned

`t`	Trace color was updated.
`nil`	Trace color could not be updated.

Example

```
lntSetTraceColor(1 "cyan" window(2))
```

Updates the color of the trace with ID `1` in window `2` to color `cyan`.

Related Topics

Tracing Nets

Net Tracer Options

lntSetTraceVisibility

```
lntSetTraceVisibility(
     x_traceID
     x_visibility
     [ w_window ]
     )
     => t / nil
```

Description

Sets the visibility of the specified trace in the specified window.

Arguments

x_traceID	ID of the trace for which visibility is to be updated.
x_visibility	Visibility to be set for the trace. Type `0` to make the trace invisible or `1` to make it visible.
w_window	ID of the window containing the specified trace. When not specified, the trace (if it exists) is updated in the current window.

Value Returned

`t`	Trace visibility was updated in the specified window.
`nil`	Trace visibility could not be updated.

Example

Makes the trace with ID `1` in window 2 invisible.

```
lntSetTraceVisibility(1 0 window(2))
```

Related Topics

Tracing Nets

lntShowHideAllTraces

```
lntShowHideAllTraces(
     x_showHide
     [ w_window ]
     )
     => t / nil
```

Description

Shows or hides all traces in the cellview contained in the specified window.

Arguments

x_showHide	Integer specifying whether traces are to be shown or hidden. Type `0` to hide all traces or `1` to show all traces.
w_window	ID of the window containing the cellview in which traces are to be shown or hidden. When not specified, the trace (if it exists) is updated in the current window.

Value Returned

`t`	All traces in the specified window were either shown or hidden.
`nil`	The operation was unsuccessful.

Example

```
lntShowHideAllTraces(0 win)
```

Hides all traces in the specified window.

```
lntShowHideAllTraces(1)
```

Shows all traces in the current window.

Related Topics

Tracing Nets

lntShowNeighbors

```
lntShowNeighbors(
     w_window
     )
     => t / nil
```

Description

Displays computed neighbors for the specified window. The computed neighbors can be hidden using the `lntHideNeighbors` SKILL function. Use `lntClearNeighbors` to delete all neighbors.

Arguments

w_window	ID of the window in which computed neighbors are to be shown.

Value Returned

`t`	Computed neighbors in the specified window were shown.
`nil`	The window has no computed neighbors.

Example

```
window = hiGetCurrentWindow()
lntComputeNeighbors(window 0.04)
lntHideNeighbors(window)
lntShowNeighbors(window)
```

Computes neighbors for the traces in the current window, hides them, then shows them again.

Related Topics

lntHideNeighbors

lntClearNeighbors

Tracing Nets

Finding Neighboring Shapes of a Traced Shape

Net Tracer Neighbors Form

lxAbutGetNeighbors

```
lxAbutGetNeighbors(
     { ld_instId | d_instId }
     [ ?direction { 'up | 'down | 'left | 'right | 'horizontal | 'vertical | 'all } ]
     [ ?depth { 0 ... INT_MAX } ]
     [ ?stopGroup d_stopGroup ]
     [ ?select { t | nil } ]
     )
     => ld_instId / nil
```

Description

Returns the list of instance IDs found by traversing from an instance list, or an instance, through the associated abutment groups, in the supplied direction.

The traversal can be restricted by providing a maximum traversal depth. A value of `0` or `nil` indicates no restriction. If an abutment group object is supplied via the `?stopGroup` argument, the group is not traversed. The `?select` argument, when `true`, selects the returned instances.

Arguments

`ld_instId | d_instId`

List of instance IDs, or a single instance ID.

`?direction`

The direction in which the abutment groups are traversed.

The default is `'all`.

`?depth`

Integer specifying the maximum depth of traversal. If set to `0` or `nil`, the traversal is unrestricted.

The default is `0`.

`?stopGroup d_stopGroup`

Database ID of the abutment group object that should not be traversed.

`?select`

Selects the returned instances when set to `t`.

Value Returned

ld_instId	The list of abutted instances found.
`nil`	No abutted instances were found.

Example

```
lxAbutGetNeighbors( inst1 ?direction 'right ?depth 5 ?stopGroup group1 )
```

lxChain

```
lxChain(
     l_insts
     l_point
     [ ?groups l_groups ]
     [ ?preserveExistingChains { t | nil } ]
     [ ?useDeviceOrder { t | nil } ]
     [ ?interdigitateChains { t | nil } ]
     [ ?mirror { t | nil } ]
     [ ?permute { t | nil } ]
     [ ?allowSingleBulk { t | nil } ]
     [ ?dummyFlexBothEndNets { t | nil } ]
     [ ?syncChains { t | nil } ]
     [ ?mirrorEquivOrients { t | nil } ]
     [ ?preserveRows { t | nil } ]
     [ ?useAbutSpacing { t | nil } ]
     [ ?maxChainSize x_maxChainSize ]
     [ ?contacts { 0 | 1 | 2 } ]
     [ ?chainAlignPMOS { Top | Center | Bottom } ]
     [ ?chainAlignNMOS { Top | Center | Bottom } ]
     [ ?chainLeftNet { Source | Drain | Either } ]
     [ ?abutStrategy { sdFirst | dummyFirst } ]
     [ ?optimize { Tracks | Abutments | Both } ]
     [ ?multiRow { NP | PN | NPPN | PNNP } ]
     [ ?effort { Minimal | Nominal | Maximal } ]
     )
     => l_chains / nil
```

Description

Chains a list of instances and places the generated chains at the specified position in the layout. Using the optional arguments will override the existing settings of the environment variables: `chainPreserveExistingChains`, `chainUseDeviceOrder`, `lxAllowPseudoParallelNets`, `chainMirror`, `chainPermutePins`, `chainAllowSingleBulk`, `chainDummyFlexBothEndNets`, `chainSyncChains`, `chainMirrorEquivOrients`, `chainPreserveRows`, `chainUseAbutSpacing`, `lxChainAlignPMOS`, `lxChainAlignNMOS`, `chainLeftNet` and `chainAbutStrategy`.

Arguments

l_insts	Database IDs of the instances to be chained.
l_point	Defines the position of the first instance in the generated chain. For more information about this argument, see Positioning of Chains.
?groups	List of groups where each group is a list of instances. Builds a group chain for each group. The group chains can be mirrored and are always preserved. The default is `nil`.
?preserveExistingChains	Prevents existing chains from being broken. The default value is `nil`. If `useDeviceOrder` and `preserveExistingChains` arguments are both set, the SKILL function maintains the existing abutments.

`?useDeviceOrder`

Maintains the relative starting positions of the specified instances when forming the chain. Instances are sorted in XY order of their origins (from lowest to highest) and the resultant list used to abut the devices from right to left.

The default value is `nil`.

Note:

- If `useDeviceOrder` is set to `t`, devices are chained even if their S/D pins are on different nets. In this case, the devices are abutted on dummy pins. In addition, when all the advanced node devices have the same bulk, all the individual chains created using these devices are merged into a single chain.

- If `useDeviceOrder` is set to `t`, the `lxChain()` function supports abutment of instances that have different values set for the `abutClass` property but have their `abutClass` values defined as equivalent. If `useDeviceOrder` is set to `nil`, these `equivalentAbutClasses` are ignored.

- If `useDeviceOrder` and `preserveExistingChains` arguments are both set, the SKILL function maintains the existing abutments.

`?interdigitateChains`

Identifies and defines the pseudo parallel nets.

The default value is `nil`.

`?mirror`

Mirrors instances.

The default value is `t`.

For more information about the behavior of this argument, see Mirroring and Permutation.

`?permute`

Permutes instances.

The default value is `t`.

For more information about the behavior of this argument, see Mirroring and Permutation.

`?allowSingleBulk`

Abuts devices if only one of the devices has a bulk and the `?useDeviceOrder` argument is set to `t`.

The default value is `t`.

Note: The value of the `?allowSingleBulk` argument defaults to the value of the `chainAllowSingleBulk` environment variable.

`?dummyFlexBothEndNets`

(IC6.1.8 Only) Controls whether the dummy devices can have different source/drain connectivity.

The default value is `t`.

`?syncChains`

Synchronizes chains by abutting the specified instances by columns. The instances are first arranged into rows and then abutted row-wise from right to left in each column pair.

- For columns that have no instances in the row being abutted, abutment does not take place. Such columns are spaced out.
- When set to `nil`, instances are abutted by row, irrespective of the number of instances in each row.

The `?syncChains` argument uses the row bBox-Y overlaps to determine the rows that need to considered for synchronizing the chains.

The `?syncChains` argument does not support groups. However, in case of a single row, groups are supported by calling the non-syncChains algorithm with `?useDeviceOrder` set to `t` if groups exist, and the number of rows is `1`.

The value of the `?syncChains` argument defaults to the value of the `chainSyncChains` environment variable. By default, the value of `chainSyncChains` is `nil`.

For more information, see Synchronized Chaining.

`?mirrorEquivOrients`

Mirrors to equivalent orientations. which means that *Chaining* mirrors `R0` to `MY` or vice versa.

The default value is `t`.

When set to `nil`, *Chaining* mirrors `MX` to `R180` and vice versa.

`?preserveRows`

Accepts multiple rows as input and chains each row using the optional parameters supplied and with `useDeviceOrder` set.

- If flat instance list is supplied, the instances are sorted into rows.
- If list of rows is supplied, the rows are used as specified.

Note: The supplied position defines the lower left of the bottom row.

`?useAbutSpacing`

Controls how chains are spaced.

If set to `t`, chains are spaced using the abutment spacing value. If set to `nil`, chains are spaced using the value of the `lxPositionMinSep` environment variable.

The default value is `nil`.

`?maxChainSize` *`x_maxChainSize`*

Defines the maximum chain size, specified using any integer value between `1` to *`INT_MAX`*.

The default value is *`INT_MAX`*.

`?contacts`

Controls whether abutment will delete or keep contacts.

- `0` means abutment will automatically determine if contacts should be deleted or kept.
- `1` means abutment will delete contacts.
- `2` means abutment will keep contacts.

The default value is `0`.

`?chainAlignPMOS`

Controls PMOS chaining alignment using values: `"Top"`, `"Center"`, and `"Bottom"`. The values should be enclosed in quotation marks.

The default value is `"Top"`.

`?chainAlignNMOS`

Controls NMOS chaining alignment using values: `"Top"`, `"Center"`, and `"Bottom"`. The values should be enclosed in quotation marks.

The default value is `"Bottom"`.

`?chainLeftNet`

Controls whether source or drain nets are optimized to the left of the generated chains. Set to `"Either"` if you have no preference (and to maintain the default behavior from previous releases). The values should be enclosed in quotation marks.

The default value is `"Source"`, which means the generated chain is optimized so that one of its source nets is on the left-hand side of the chain if possible.

Note:

- Source and drain refer to schematic source and drain nets, not layout source and drain nets, which may have been permuted.
- `?chainLeftNet` is an optional argument, which, if set, will override the existing setting of the chainLeftNet environment variable.

`?abutStrategy`

Abuts on source/drain or dummy shapes first.

The default value is `sdFirst`.

- If set to `sdFirst`, abuts on source/drain first. If this fails, tries abutting on dummy shapes.
- If set to `dummyFirst`, abuts on dummy shapes first. If this fails, tries abutting on source/drain.

`?optimize`

Optimizes the chains by minimizing the number of tracks or maximizing the number abutments or both.

The default value is `"Both"`.

`?multiRow`

Controls the row order from bottom to top for multi-row placement.

The default is `""`.

`?effort`

Controls the effort applied during chaining. Increased effort can improve the quality of results, but it can be time-consuming.

The default is `Nominal`.

Value Returned

`l_chains`	List of chains generated where each chain is identified as a list of database IDs.
`nil`	The specified instances were not chained.

Tip

All arguments that take a string value must have their values enclosed in quotation marks.

Examples

Example 1

```
lxChain(list(p0 p1 p2) 0:0 ?useDeviceOrder t ?mirror t ?permute t ?chainAlignPMOS
"Center" ?chainLeftNet "Source")
```

Chains devices “|P0”, “|P1” and “|P2” in XY order with odd devices mirrored and even devices permuted. The generated chains are placed at position 0:0 and are center-aligned with source nets optimized to the left.

Example 2

```
lxChain(list(n0 n1 n2) 5:5 ?preserveExistingChains t ?mirror nil ?permute t
?chainAlignNMOS "Top" ?chainLeftNet "Drain")
```

Chains devices “N0”, “|N1” and “|N2” with all devices permuted and existing chains preserved. The generated chains are placed at position 5:5 and are top-aligned with drain nets optimized to the left.

Example 3

```
lxChain(list(p0 p1 p2 n0 n1 n2) 10:10 ?useDeviceOrder t ?mirror t ?permute nil
?multiRow "NPPN" ?optimize "Abutments" ?effort "Maximal")
```

Chains devices “|P0”, “|P1”, “|P2” “N0”, “|N1” and “|N2” in XY order with odd devices mirrored and even devices not permuted. The generated chains are placed in multi-row pattern NPPN with the bottom row at position 10:10. Abutments are maximized with effort set to maximal for best quality of results.

Example 4

```
lxChain(geGetSelSet() 0:0 ?preserveRows t)
```

Sorts the selected set into row order and then chains each row using the default optional parameters with `useDeviceOrder` set. The lower left of the bottom row is positioned at `0:0`.

```
lxChain(list(list(I0 I1) list(I2 I3 I4)) 0:0 ?preserveRows t)
```

Chains rows (`I0 I1`) and (`I2 I3 I4`) using the default optional parameters with `useDeviceOrder` set. The lower left of `I2` is positioned at `0:0`.

Example 5

```
lxChain(list(p0 p1 p2) 0:0 ?useAbutSpacing t)
```

Chains the named devices and spaces each chain using the abutment spacing value.

Example 6

```
lxChain(geGetSelSet() 0:0 ?useDeviceOrder t ?groups list(list(dbId1 dbId2)
list(dbId3 dbId4 dbId5)))
```

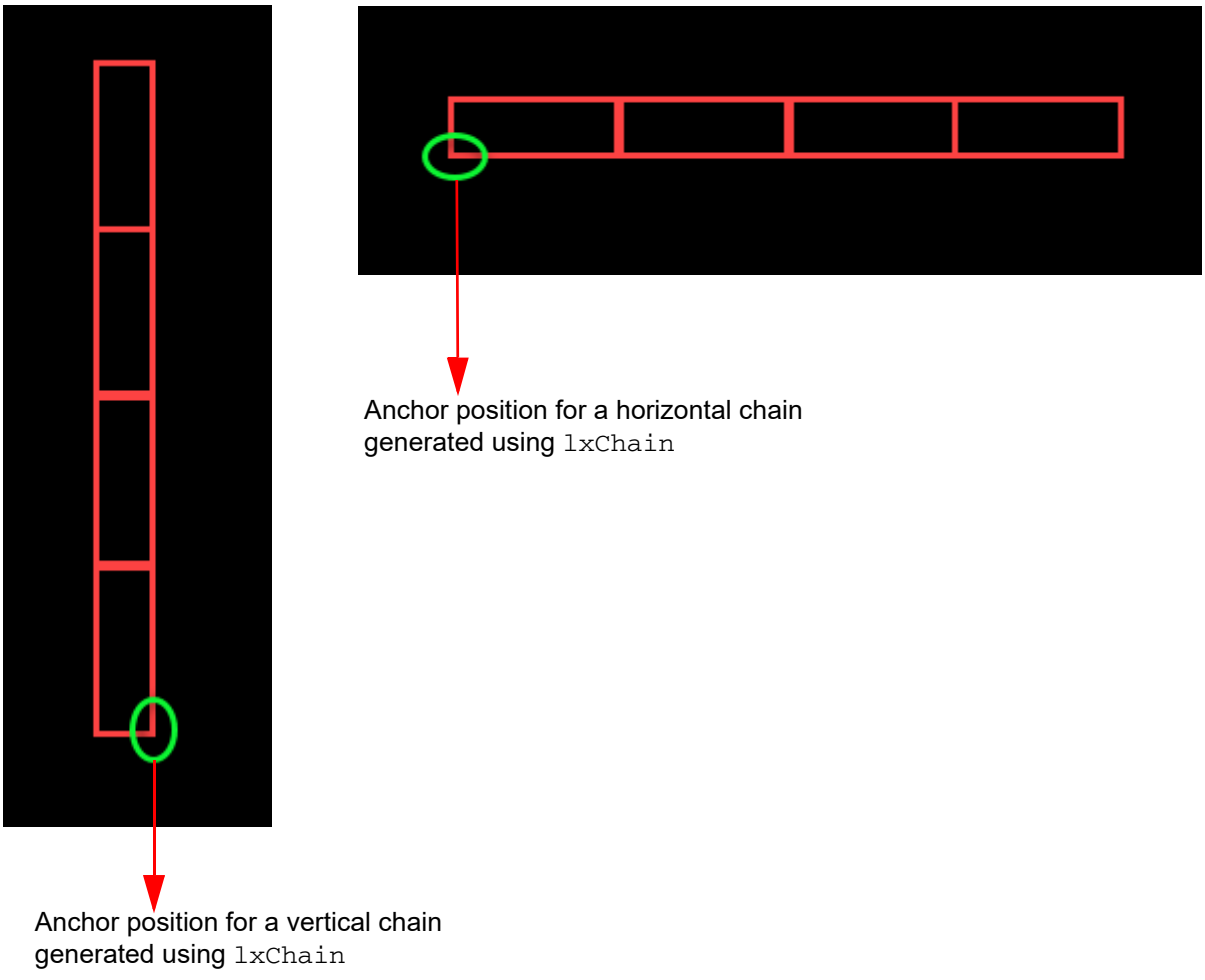
Chains a list of instances into a group.

Additional Information

Positioning of Chains

The `lxChain` SKILL function uses the *`l_point`* argument to position the first instance of the generated chains.

- For horizontal devices, the position of the first instance of the chain is the lower left-corner of the bbox, as illustrated in the figure below.
- For vertical devices, the position of the first instance of the chain is the lower right-corner of the bbox, as illustrated in the figure below.



Mirroring and Permutation

For mirroring and permutation, `lxChain` depends on the state of the `?mirror` and `?permute` arguments:

- If both `?mirror` and `?permute` arguments are set to `t`, odd devices are mirrored and even devices are permuted.
- If the `?mirror` argument is set to `t` and the `?permute` argument is set to `nil`, odd devices are mirrored but even devices are not permuted.
- If the `?mirror` argument is set to `nil` and the `?permute` argument is set to `t`, all devices are permuted.
- If both the arguments are set to `nil`, no devices are mirrored or permuted.

Synchronized Chaining

Synchronized chaining using the `syncChains` argument can help make a design more routable by allowing each column to be aligned on the same net for all the rows. Abutment in this case is performed in column pairs, not rows—abutting the specified instances from right to left, starting from the bottom most row in the column pair to the top most row, and using the specified `mirror` and `permute` options.

The abutment begins with looking for the common net that applies to instances in all the rows within the column pair.

If the common net is found:

- Abutment is performed using the supplied `sdFirst` or `dummyFirst abutStrategy`.
    - If the common net is not found, abutment is performed using the `dummyFirst abutStrategy`.
- (IC6.1.8 Only) Abutment is performed using the `sdFirst abutStrategy`.
    - If the common net is not found, abutment is performed on any net.

Note: For a column pair that abuts, the contacts are consistently either dropped or kept for a column. Therefore, in a column pair, you can have one column with contacts dropped and the other with contacts kept. But, you cannot have a column that has one pair with contacts dropped and the other with contacts kept.

For situations, where any column pair cannot abut on any net and abutment using the `dummyFirst abutStrategy` is not possible, the column pair is unabutted.

Example: SyncChaining

```
lxChain(list(list(dbId1, dbId2) list(dbId3, dbId4, dbId5)) 0:0 ?syncChains t)
where,
row1 = list(dbId1, dbId2)
row2 = list(dbId3, dbId4, dbId5)
```

Illustrates how the rows are used, if the list of rows is specified.

```
lxChain(list(dbId1, …, dbIdN) 0:0 ?syncChains t)
```

Illustrates how the instances are sorted into rows using the Y co-ordinate of the instance origin, if the flat instance list is specified.

Chaining Photonic Waveguide Instances and Top-Level Photonic Pins

To use the `lxChain` SKILL function for chaining waveguide instances and top-level photonic pins, you must have the `Virtuoso_Photonics_Option` license checked out to run the Virtuoso Photonics Solution.

None of the optional arguments of the `lxChain` SKILL function are supported for the Virtuoso Photonics Solution. For information on obtaining the required license, contact your local Cadence representative.

Related Topics

Layout XL Environment Variables

lxChainAnchor2D

```
lxChainAnchor2D(
     anchors
     )
     => l_chains / nil
```

Description

Specifies the pins or instances to be used as anchors to derive connectivity information to chain devices until no further chains can be formed.

Arguments

anchors	A single pin or instance or a set of pins or instances to be used as anchors for generating chains.

Value Returned

l_chains	List of generated chains where each chain is identified as a list of database IDs.
nil	The specified pins or instances were not chained.

Example

```
lxChainAnchor2D(anchors)
```

Creates a chain based on the specified anchors.

lxCheck

```
lxCheck(
     d_layCV
     [ ?schCV d_schCV ]
     [ ?checkDevices { t | nil } ]
     [ ?checkExtractLayout { t | nil } ]
     [ ?extractStopLevel [0-32] ]
     )
     => 0 - 100 / -1
```

Description

Checks the layout for XL compliance and reports binding status in an Info pop-up box. It is recommended to use the `lxCheck` SKILL function to look for any possible problems in the design if the Navigator *XL Status* column is not clean. After the issues reported by `lxCheck` are resolved, running the *Connectivity – Update Binding* command should give an improved XL Status.

Arguments

d_layCV	Database ID of the layout cellview.
?schCV *d_schCV*	Database ID of the schematic cellview.
?checkDevices	Checks the lower-level cellviews in the layout for opens and shorts when extracted to `extractStopLevel`.
?checkExtractLayout	Controls the extraction of layout nets for the layout being checked for XL-compliances issues.
?extractStopLevel	Specifies the hierarchy depth for extraction. The default is `0`. The extraction depth must be an integer between `0` and `32`.

Value Returned

`0 - 100`	The average compliance.
`-1`	The check failed.

Example

```
schCV=dbOpenCellViewByType( schLibName schCellName schViewName )
layCV=dbOpenCellViewByType( layLibName layCellName layViewName )
lxCheck(layCV ?schCV schCV ?checkDevices 50)
```

Additional Information

You can also trigger an XL-compliance check for the layout by choosing the *Connectivity – Check – XL Compliance* command in the Layout XL window. The command calls the `lxCheck` SKILL function to run the various layout checks.

Related Topics

Design Check for Layout XL Compliance.

lxCheckAgainstSource

```
lxCheckAgainstSource(
     d_schCellViewID
     d_layCellViewID
     [ ?masterDiff { t | nil } ]
     [ ?paramDiff { t | nil } ]
     [ ?unboundInsts { t | nil } ]
     [ ?connectivity { t | nil } ]
     [ ?maxDiffsLimit x_maxDiffsLimit ]
     [ ?logFileName t_logFileName ]
     [ ?appendToLog { t | nil } ]
     [ ?nameDiff { t | nil } ]
     [ ?dummyDiff { t | nil } ]
     [ ?busTermDiff { t | nil } ]
     [ ?virtualHierDiff { t | nil } ]
     )
     => t / nil
```

Description

Generates a report highlighting the differences between the schematic and the layout cellview. If you use the optional arguments, they override the existing environment variable settings in the `.cdsenv` file.

Arguments

d_layCellViewID	Database ID of the target layout cellview.
d_schCellViewID	Database ID of the source schematic cellview.
masterDiff	Reports instance masters that are missing or mismatched between the layout and the schematic.
?paramDiff	Reports CDF parameter mismatches and property mismatches between the schematic and layout views.
?unboundInsts	Reports layout instances that are not bound to schematic instances.
?connectivity	Reports connectivity mismatches on top level pins and global nets; mismatched or missing terminals and instance terminals; and unbound nets in the layout.
?maxDiffsLimit	Limits the number of differences reported in the CIW for each of the four categories. For example, if you set the maximum number of differences to 100, the command reports up to a maximum of 100 cellview master differences, 100 unbound instance differences, and so on. Any messages over the specified limit are suppressed in the CIW but are still listed in the log file.
?logFileName	Specifies the name of the file in which all the *Check Against Source* messages are written.
?appendToLog	Appends the results of the current run to the specified log file instead of overwriting the log file.
?nameDiff	

Reports name differences for bound instances, if the Layout XL naming convention does not match that of the schematic. The name differences are updated by *Update Components And Nets*.

The default is `nil`.

Example:

If the `nameDiff` argument is set to `t`, and the schematic instance `INV/N0` is bound to the layout instance, `I0`, the layout instance name is updated to `INV|N0`. If the `prefixLayoutInstNamesWithPipe` environment variable is set to `t`, the layout instance name is prefixed with an OR (`|`) bar, as `|INV|N0`.

`?dummyDiff`

Reports layout dummy instances that may require backannotation to the schematic.

The default is `nil`.

`?busTermDiff`

Reports implicit bus terminal mismatches between the layout and the schematic.

The default is `t`.

`?virtualHierDiff`

Reports mismatches in the virtual hierarchy between the layout and the schematic.

The default is `nil`.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Example

```
lxCheckAgainstSource(scv lcv)
```

Generates a *Check Against Source* report with the default environment variable settings.

```
lxCheckAgainstSource(scv lcv ?masterDiff t ?paramDiff nil ?unboundsInsts nil
?connectivity t ?logFileName "./checkReport.log")
```

Appends to `checkReport.log` the master and connectivity differences that are reported during a *Check Against Source* run.

lxCheckAndUpdate

```
lxCheckAndUpdate(
     t_libNameOrLayCV
     [ ?schCV d_schematicCellview  ]
     [ ?update g_update ]
     [ ?app t_appName ]
     [ ?layoutComplete g_layoutComplete ]
     [ ?binder t_binderValue ]
     [ ?svdbPath t_lvsSvdbPath ]
     [ ?svdbScale n_lvsScale ]
     [ ?extract g_Extract ]
     [ ?extractStopLevel n_stoplevel) }  ]
     [ ?onlyCheckUpdate g_onlyCheckUpdate  ]
     [ ?allCellsInDesign g_allCellsInDesign  ]
     [ ?allCellsInLibrary g_allCellsInLibrary ]
     [ ?excludeCells t_excludeCells  ]
     [ ?includeCells t_includeCells  ]
     [ ?leafCells t_leafCells  ]
     [ ?withoutSchematic g_cellsWithoutSchematic  ]
     [ ?filterMessage l_filterMessage ]
     [ ?createHTML g_createHTMLReport ]
     [ ?openHTML g_openHTMLReport ]
     [ ?logFile g_logFile  ]
     [ ?logFileDirectory t_logFileDirectoryPath  ]
     [ ?detailedReturn g_detailedReturn  ]
     [ ?report { t | nil }   ]
     [ ?reportOptions { t| nil }  ]
     [ ?graphical { t| nil }   ]
     [ ?targetWindow { w_windowId | nil }  ]
     )
     => nil | l_testResult | n_percentageClean
```

Description

Runs the application readiness check and updates the specified cellviews for the specified applications.

Arguments

t_libNameOrLayCV Name of the target library or layout cellview to check and update.

?schCV *d_schematicCellview*

Specifies the schematic cellview to use as the layout connectivity reference.

`?update` *`g_update`*

Specifies whether to check or update the specified application.

`?app` *`t_appName`* Name of the application to check and update.

`?layoutComplete` *`t_layoutComplete`*

When set to `t`, performs hierarchical rebinding and clears logical connectivity. Unbound or ungenerated instances, terminals, or nets are not updated.

LVS binding is only allowed when `?layoutComplete` is `t`.

The default is `nil`.

`?binder` *`t_binderValue`*

Specifies the type of binding to perform during application update.

Valid values are:

`Current` - Use the existing binding.

`Rebind` - Update using the *Update Binding* command.

`LVS` - Rebind using the binding information in the svdb directory.

The default is `Rebind`.

`?svdbPath` *`t_lvsSvdbPath`*

Specifies the path to the PVS svdb directory containing:

`xyz``.ixf` instance cross-reference file.

`xyz``.net` extracted layout CDL netlist file.

`?svdbScale` *`n_lvsScale`*

Input scale. The PVS rules file `input_scale` is required only if `input_scale` is different from the layout DBUPerUU.

`?extract` *`g_Extract`* Specifies whether the layout needs to be extracted.

The default is `nil`.

`?extractStopLevel` *`n_stoplevel`*

Specifies the hierarchy depth for extraction.

The default is `-1`.

`?onlyCheckUpdate` *`g_onlyCheckUpdate`*

Specifies a check or update criteria to ensure only those checks or updates that match the path are performed.

Example**:**

```
list("Unbound") or list("Unbound" "Instances")
```

`?allCellsInDesign` *`g_allCellsInDesign`*

Checks and updates all cells in the specified cellview.

`?allCellsInLibrary` *`g_allCellsInLibrary`*

Checks and updates all cells in the specified library.

`?excludeCells` *`t_excludeCells`*

List of cells to be excluded when the `?allCellsInDesign` or `?allCellsInLibrary` option is specified.

Example:

```
(\\*pdk\\* via\\* layout) (STD\\* \\*FILLER\\* layout\\*)
```

`?includeCells` *`t_includeCells`*

List of cells to be included when the `?allCellsInDesign` or `?allCellsInLibrary` option is specified.

Example:

```
(DESLIB TOP layout) (DES\\* \\*MYCELL\\* layout\\*)
```

`?leafCells` *`t_leafCells`*

List of cells to be treated as leaf cells. Only the terminals of leaf cells are checked when `?allCellsInDesign` or `?allCellsInLibrary` option is specified.

`?withoutSchematic` *`g_cellsWithoutSchematic`*

Checks and updates layout cells without a schematic connectivity reference.

The default is `nil`.

`?filterMessage` *`l_filterMessage`*

List of messages to be filtered from the Check and Update report.

Example

```
LX-1946 BND-1105
```

`?createHTML` *`g_createHTMLReport`*

Specifies whether to create an HTML report.

The default is `t`.

`?openHTML` *`g_openHTMLReport`*

Specifies whether to open the HTML report in the default browser.

The default is `nil`.

`?logFile` *`g_logFile`* Specifies whether to save the report to a text log file.

The default is `nil`.

`?logFileDirectory` *`t_logFileDirectoryPath`*

Specifies a directory for the log file.

`?detailedReturn` *`g_detailedReturn`*

Specifies whether the return value should provide a detailed list of checks including information such as `Passes`, `Failed`, and `Needs attention`.

Example:

```
list(list("Direction"   "Passed")
list("Name - Nets" "Passed")
list("Parameter"   "Needs attention")
list("Signal type" "Passed"))
```

`?report { t | nil }` Specifies whether the check and update report is output to the CIW or HTML report.

The default is `t`.

`?reportOptions { t| nil }`

Specify whether options are output to the CIW or an HTML report.

The default is `t`.

`?graphical { t| nil }`

When set to `t`, extra checks for read-only layout or schematic extraction are run that might also display a dialog box.

The default is `nil`.

`?targetWindow { w_windowId | nil }`

Specifies whether to display a progress bar.

The default is `nil`.

Value Returned

`l_testResult`	List of checks, `Passed`, `Failed`, and `Needs attention`, if a single cell and single application is tested.
	Example
	`list(list("Direction" "Passed")`
	`list("Name - Nets" "Passed")`
	`list("Parameter" "Needs attention")`
	`list("Signal type" "Passed"))`

`n_percentageClean`	Pass percentage for the tests: `%Clean 0-100`
`nil`	The operation failed.

Examples

Checks schematic versus layout. This is the default check.

```
lxCheckAndUpdate(lcv)
```

Checks schematic versus layout and updates the layout.

```
lxCheckAndUpdate(lcv ?update t)
```

Checks the Design Planner application.

```
lxCheckAndUpdate(lcv ?app "Design Planner")
```

Checks and updates the schematic v layout application and checks the Design Planner application.

```
lxCheckAndUpdate(lcv ?app list(list("Schematic v Layout" t) list("Design Planner"
nil))
```

Checks all cellviews in the design.

```
lxCheckAndUpdate(lcv ?allCellsInDesign t)
```

Checks all cells in the library.

```
lxCheckAndUpdate("DESIGN_LIB")
```

lxCheckAndUpdateRegister

```
lxCheckAndUpdateRegister(
     t_appName
     g_path
     g_check
     g_update
     [ ?severity t_severity ]
     [ ?object t_objectType ]
     [ ?checkFunction t_checkFunction ]
     [ ?updateFunction t_updateFunction ]
     [ ?configCB t_arcCallBackFunction ]
     [ ?master g_callCheckUpdateMaster ]
     [ ?subMaster g_callCheckUpdateSubmaster ]
     [ ?markers g_createMarkers ]
     )
     => t / nil
```

Description

Registers any user-defined SKILL check and update functions needed for the application readiness check.

Arguments

t_appName	Application name for which to add user-defined check and update functions.
g_path	Path where the option is in the form.
g_check	Specifies the default value of the user-defined check. Valid values: `t`, `nil`, `""` The empty string `""` means, there is no a check available.
g_update	Specifies the default value of update. Valid values: `t`, `nil` The empty string `""` means, there is no update available.
`?severity` *t_severity*	Specifies the severity level of any messages generated by the check. Valid values: `"Error"`, `"Warning"`, `"Info"` The default is `"Error"`

`?object` *`t_objectType`*

Object type to pass to the check and update functions.

Valid values: `"Inst"`, `"Term"`, `"Net"`,`"Cellview"`, or multiple types, for example `"Inst Net"`

The default is `"Inst"`.

`?checkFunction` *`t_checkFunction`*

Name of the check function.

The default is `""`, which means that no check function is defined.

`?updateFunction` *`t_updateFunction`*

Name of the update function.

The default is `""`, which means that no update function is defined.

`?configCB` *`t_arcCallBackFunction`*

The callback function to be called from the *Configure* column of the Application Readiness Checker form.

The default is `""`.

`?master` *`g_callCheckUpdateMaster`*

Specifies whether to call check and update once per master cellview.

The default is `nil`.

`?subMaster` *`g_callCheckUpdateSubmaster`*

Specifies whether to call check and update once per submaster cellview.

The default is `nil`.

`?markers` *`g_createMarkers`*

Specifies whether to create markers.

The default is `t`.

Value Returned

`t`	The specified check and update functions are registered.
`nil`	The operation failed.

Examples

The example first specifies the callback functions `layoutViewNameCheck` and `layoutViewNameUpdate` and then registers them using `lxCheckAndUpdateRegister`.

The `getHierPath` function is not defined here. This function returns the hierarchy path as a string.

Define the function `layoutViewNameCheck` to check that every layout instance uses the layout view and reports if any views such as `layout1` and `layoutTmp` are being used.

```
procedure(layoutViewNameCheck(srcPaths tgtPaths update)
  let((res msg cv inst pathName)
    res = tconc(nil nil)
    foreach(path tgtPaths
      inst = car(last(path))
      unless(inst~>viewName == "layout"
        if(update then
          master = dbOpenCellViewByType(inst~>libName inst~>cellName "layout")
          when(master
            dbRemasterAnyInst(inst master)
          )
        else
          pathName = getHierPath(path nil)
          sprintf(msg "Layout view name - instance %s \t%s/%s" pathName
inst~>cellName inst~>viewName)
          tconc(res list(car(path) msg))
        )
      )
    )
    cdar(res)
  )
)
```

Define the check and update functions:

```
procedure(layoutViewNameCheck(srcPaths tgtPaths)
 layoutViewName(srcPaths tgtPaths nil)
)
procedure(layoutViewNameUpdate(srcPaths tgtPaths)
 layoutViewName(srcPaths tgtPaths t)
)
```

Register the `layoutViewNameCheck` and `layoutViewNameUpdate` functions using `lxCheckAndUpdateRegister`.

```
lxCheckAndUpdateRegister(
    "Schematic v Layout"                  ; app
        list("Custom" "Layout view name") ; path
        t                                 ; default value for check
        t                                 ; default value for update
        ?object         "Inst"
        ?checkFunction  "layoutViewNameCheck"
        ?updateFunction "layoutViewNameUpdate"
        ?severity       "Error"
        ?markers        t)
```

lxCheckAndUpdateRemove

```
lxCheckAndUpdateRemove(
     t_appName
     [ g_path ]
     )
     => t / nil
```

Description

Removes the specified application or test from the application readiness checks.

Arguments

t_appName	Name of the application to remove from checks and updates.
g_path	Path from where to remove checks and updates. If the path is not `nil`, all checks and updates for the specified application are removed from below the path. If the path is `nil`, the specified application is removed. The default is `nil`.

Value Returned

`t`	The specified application or test is removed from the *Check* and *Update* tests run from the application readiness checks.
`nil`	The operation failed.

Examples

The Design Planner application is removed and will not be available to run from the Application Readiness Checker form or the `lxCheckAndUpdate` SKILL function.

```
lxCheckAndUpdateRemove("Design Planner")
```

The *Signal* type check and update is removed from the *Schematic v Layout* application.

```
lxCheckAndUpdateRemove("Schematic v Layout" list("Signal type"))
```

lxCheckAndUpdateSet

```
lxCheckAndUpdateSet(
     t_appName
     g_path
     ?check g_check
     ?update g_update
     ?severity g_severity
     )
     => t / nil
```

Description

Sets check, update, and severity values for the Application Readiness Checker form.

Arguments

t_appName	Name of the application for which to set check, update, and severity values.
g_path	Path from where to set check, update, and severity values. If the path is not `nil`, the check, update, and severity values are set on all checks and updates below the specified path. If the path is `nil`, the check, update, and severity values are set on all checks and updates for the application. The default is `nil`.
`?check` *g_check*	Specify whether the check value should be set. Valid values: `t`, `nil`, `""` The empty string `""` means the check value is not set.
`?update` *g_update*	Specify whether the update value should be set. Valid values: `t`, `nil`, `""` The empty string `""` means the update value is not set.
`?severity` *g_severity*	Specifies the severity level of any messages generated by the check. Valid values: `"Error"`, `"Warning"`, `"Info"` The default is `"Error"`.

Value Returned

`t`	Values for check, update, and severity were set successfully.
`nil`	The set operation failed.

Examples

For the Schematic v Layout application, checks for unbound instances, terminals, and nets, sets severity level to "`Error`", but does not perform any updates to correct the issues.

```
lxCheckAndUpdateSet("Schematic v Layout"
                        list("Unbound")
                        ?check    t
                        ?update   nil
                        ?severity "Error")
```

For Schematics v Layout application, check for unbound, and nets and sets severity level to `"Warning"`.

```
lxCheckAndUpdateSet("Schematic v Layout"
                    list("Unbound" "Nets")
                    ?severity "Warning")
```

lxCheckLib

```
lxCheckLib(
     libName
     [ ?checkBindings { t | nil } ]
     [ ?checkExtractLayout { t | nil } ]
     [ ?summaryLogFileName t_summaryLogFileName ]
     [ ?extractStopLevel [0-32] ]
     )
     => 0 - 100 / -1
```

Description

Checks the layouts in the specified library for opens and shorts when extracted to `extractStopLevel`, and displays the results in CIW and in the Info window. CAD engineers are advised to run this SKILL function to identify the instances that are likely to cause issues with Update Binding.

Arguments

libName	Name of the library in which the layouts are checked.
?checkBindings	Performs all binding related checks on all cells in the library.
?checkExtractLayout	Controls the extraction of layout nets for the library being checked.
?hierSummaryLogFileName	Specifies the name of the file to which the Summary is written, if the binding check is performed.
?extractStopLevel	Specifies the hierarchy depth for extraction when checking for opens and shorts, The default is `0`. The extraction depth must be an integer between `0` and `32`.

Value Returned

`0 - 100`	The average compliance.
`-1`	The check failed.

Example 1

```
lxCheckLib (libName)
```

Example 2

```
lxCheckLib("PARTIAL" ?checkBindings t ?summaryLogFileName "./checkLibPARTIAL.log"
?extractStopLevel 1)
INFO (LX-1935): Starting to check library 'lib'.
INFO (LX-1939): Finished checking library 'lib'.
30
```

Checks a library named PARTIAL, performing all binding checks on all the cells in the library. No shorts are found. Average compliance is 30.%. The report is printed to the specified log file.

Example 3

```
lxCheckLib("lib")
INFO (LX-1935): Starting to check library 'lib'.
INFO (LX-1936): Failed to create an instance with master 'lib/cellA/layout'.
INFO (LX-1937): Failed to identify shorts between terminals because the cellview
'lib/cellB/layout' has no terminals.
INFO (LX-1938): Cellview 'lib/cellC/layout' has a short between terminals 'A' and
'B' at extract level 32.
INFO (LX-1939): Finished checking library 'lib'.
```

The library check failed because the cellview has no terminals.

Example 4

```
lxCheckLib("lib")
INFO (LX-1935): Starting to check library 'lib'.
WARNING (LX-3122): Cannot find library 'lib'. Ensure that the library name is
correct and that it exists on disk.
-1
```

The library could not be checked because an incorrect library name was specified.

lxCloneGetEquivalentFigs

```
lxCloneGetEquivalentFigs(
     t_figure
     )
     => l_figures / nil
```

Description

Returns a list of equivalent figures from other clones belonging to the same synchronous clone family.

Arguments

d_figure	Database ID of a figure that is member of a synchronous clone figGroup.

Value Returned

l_figures	List of equivalent figures in other clones.
nil	The operation failed because: ■ The specified figure is not a member of a synchronous figGroup or the synchronous clone figGroup has not been edited, or both. ■ Unsynchronized figures, such as labels, do not have equivalent figures.

Examples

Here, `r5` is an instance inside `clone2`.

```
r5 = dbFindAnyInstByName(cv "R5")
db:0x2dfdae21
```

Returns `nil` when the clone has not been edited.

```
lxCloneGetEquivalentFigs(r5) ;
nil
```

Edit the clone.

```
geSelectFig(clone2)
```

```
t

leHiEditInPlace()
t
```

Returns a list of figures after the clone has been edited.

```
lxCloneGetEquivalentFigs(r5)~>name
("R3" "R14")
```

Related Topics

Generating Synchronous Clones

lxCmdOptions

```
lxCmdOptions(
     )
```

Description

Implements a command option for the listed Virtuoso Layout Suite XL commands when the right-mouse button bindkey is pressed. For other Layout XL commands, the right-mouse button bindkey performs the operations as supported by `cmdOptions`.

Commands and Command Options

Chaining and *Folding* — when either of the commands is active with something to place in the canvas, each time the right mouse button is clicked, the devices to place rotate R90.

Arguments

None

Value Returned

None

lxCmdShiftOptions

```
lxCmdShiftOptions(
     )
     => nil
```

Description

Activates the bindkeys associated with `Shift`-click and `Shift`-right click during the operation of `enterFunction` commands. Typing this command in the Command Interpreter Window has no effect.

Arguments

None

Value Returned

`nil`	The bindkeys were activated.

lxComputeViaParams

```
lxComputeViaParams(
     d_viaDefID
     d_objID
     [ ds_constraintGroup ]
     [ l_params ]
     [ f_pathWidth1 ]
     [ t_pathDirection1 ]
     [ f_pathWidth2 ]
     [ t_pathDirection2 ]
     [ l_options ]
     )
     => list_of_params | nil
```

Description

Returns a list of DRC-correct via parameters for a specified standard via definition or a custom via definition that references either a `cdsVia` or a correctly-defined `syEnhContact`. The via parameters returned are: *`n_cutWidth`*, *`n_cutHeight`*, *`l_cutSpacing`* (X and Y), *`l_layer1Enc`* and *`l_layer2Enc`* (top, bottom, left, and right).

You can pass any of the listed parameters to the function using the *`params`* argument. If the parameter value you specify passes the design-rule check (DRC), `lxComputeViaParams` returns the value unchanged. If the value you specify does not pass DRC, `lxComputeViaParams` calculates the correct value and returns it instead.

If you specify the *`width`* and *`path`* parameters, a via is computed using the width and the direction of the two connecting paths. For example, if the path directions are "*`horizontal`*" and "*`vertical`*", the via is created for two crossing paths. If the directions are "*`leftToRight`*" and "*`vertical`*", the via is created for a T-junction. If the directions are "*`topToBottom`*" and "*`rightToLeft`*", the via is created for a corner. If the *`width`* and *`path`* parameters are not specified, a via is generate without a shape.

If you do not specify any parameter values, `lxComputeViaParams` takes the default value from the via definition in the technology database. If the default value passes the DRC, `lxComputeViaParams` returns it. If the default value does not pass the DRC, `lxComputeViaParams` calculates the correct parameter value and returns it.

Arguments

d_viaDefID	Database ID of the via definition for which parameters are to be calculated.
d_objID	Cellview ID of the object with which the via definition is associated.
ds_constraintGroup	Name or database ID of the constraint group.
l_params	Optional list of parameter values expressed as name-value pairs. The parameters you can specify are: "*cutWidth*", "*cutHeight*", "*cutRows*", "*cutColumns*", "*cutSpacing*","*cutSpacingX*", "*cutSpacingY*", "*layer1XEnclosure*", "*layer1YEnclosure*", "*layer2XEnclosure*", "*layer2YEnclosure*", "*layer1EncLeft*", "*layer1EncRight*", "*layer1EncTop*", "*layer1EncBottom*", "*layer2EncLeft*", "*layer2EncRight*", "*layer2EncTop*", "*layer2EncBottom*", "*originOffset*", "*justification*". If the parameter value you specify passes the design-rule check (DRC), `lxComputeViaParams` returns the value unchanged. If the value you specify does not pass DRC, `lxComputeViaParams` calculates the correct value and returns it instead.
f_pathWidth1	Width of the layer1 path in the via definition. This argument defines the width of the first of the two paths to be connected together.
t_pathDirection1	Direction of the layer1 path in the via definition. Valid values are: `leftToRight`, `rightToLeft`, `topToBottom`, `bottomToTop`, `diagRise`, `diagFall`, `horizontal`, or `vertical`. This argument defines the direction of the first of the two paths to be connected together.
f_pathWidth2	Width of the layer2 path in the via definition. This argument defines the width of the second out of the two paths to be connected together.

t_pathDirection2

Direction of the layer2 path in the via definition. Valid values are: `leftToRight`, `rightToLeft`, `topToBottom`, `bottomToTop`, `diagRise`, or `diagFall`.

This argument defines the direction of the second of the two paths to be connected together.

l_options

Optional list of parameter values expressed as name-value pairs.

The parameters you can specify are: "*calcMode*", "*calcEncMode*", "*minNumCut*", "*alignment*", "*cutBoxDirection*", "*cutClass*", "*fillOverlap*" and "*extendEncBeyondOverlap*"

For example, you can specify the parameters *t_calcMode* and *t_calcEncMode*. These parameters specify whether the via parameters or only the enclosures are to be calculated, respectively. Both these parameters can take the following values: `minRules`, `minRulesAndViaDef`, and `viaDefDefaults`. The default is `minRulesAndViaDef`. If you specify only *t_calcMode*, both *t_calcMode* and *t_calcEncMode* are set.

Other supported `l_options` parameters indicate:

- *minNumCut*: Minimum number of cuts in the via
- *alignment*: The way the via needs to be aligned. Valid values are: `auto`, `upperRight`, `upperCenter`, `upperLeft`, `centerLeft`, `lowerLeft`, `lowerCenter`, `lowerRight`, `centerRight`, and `centerCenter`
- *cutBoxDirection*: Direction of the cut box. Valid values are: `horizontal`, `vertical`, and `auto`
- *cutClass*: Name of the cut class
- *fillOverlap*: Boolean specifying whether the overlap should be filled or not. Valid Values: `t` or `nil`
- *extendEncBeyondOverlap*: Enables the via enclosures to be extended beyond the overlap region of the intersecting shapes in the *Single* (in Auto mode), *Stack* (in Auto mode), and Auto via creation modes.

Value Returned

list_of_params	List of parameters for the specified via definition. The parameters listed are specific to each type of via definition. Only those parameters that differ from the technology file defaults are listed. The parameters returned are: "*cutWidth*", "*cutHeight*", "*cutRows*", "*cutColumns*", "*cutSpacing*","*cutSpacingX*", "*cutSpacingY*", "*layer1XEnclosure*", "*layer1YEnclosure*", "*layer2XEnclosure*", "*layer2YEnclosure*", "*layer1EncLeft*", "*layer1EncRight*", "*layer1EncTop*", "*layer1EncBottom*", "*layer2EncLeft*", "*layer2EncRight*", "*layer2EncTop*", "*layer2EncBottom*", "*originOffset*", "*justification*"
nil	The *viaDefId* is not valid for the specified *objID* or *constraintGroup*), or the *viaDefId* is a customViaDef with nonstandard via parameters.

Examples

1. `cellViewId = geGetEditCellView()`
2. `viaDefs=lxGetValidViaDefs(cellViewId "highYield")`
   `; pick a viaDefId`
3. `viaId=nth(3 viaDefs)`
4. `viaParams = list(list("cutRows" 3) list("cutColumns" 3))`
5. `viaParams=append(`
   `viaParams`
   `lxComputeViaParams( viaId cellViewId "highYield"  viaParams )`
   `)`
6. `viaId=dbCreateVia(cellViewId viaId list(5 5) "R0" viaParams)`

Creates a via using the lxGetValidViaDefs, lxComputeViaParams, and dbCreateVia commands.

```
tf=techGetTechFile(ddGetObj("gpdk090"))
via=techFindViaDefByName(tf "M2_M1v")
```

```
paramList=lxComputeViaParams(via geGetEditCellView() "virtuosoDefaultSetup"
list(list("cutRows" 3) list("cutColumns" 3) list("justification" "lowerLeft")))
```

Automatically generates the cut spacing parameters for a via using lxComputeViaParams.

```
(("cutRows" 3)
    ("cutColumns" 3)
    ("cutSpacing"
                (0.2 0.2)
    )
    ("originOffset"
                (0.34 0.34)
    )
)
```

Automatically computes the via parameters and displays them in the CIW.

```
dbCreateVia(geGetWindowCellView() via 0:0 "R0" paramList)
```

Uses the computed parameters to create the associated via at `0:0`.

lxConvertSlotToPolygon

```
lxConvertSlotToPolygon(
     [ ?all { t | nil } ]
     [ ?region l_region ]
     [ ?layers l_layers ]
     )
     => t / nil
```

Description

Converts the metal shapes on the `drawing` purpose to one slotted metal polygon by creating holes where the `slot` purpose shapes cover the original metal shapes. The original metal shapes and the `slot` purpose shapes are removed.

Arguments

`?all`	Applies the command to the entire design when set to `t`.
`?region`	List of coordinates (x1 y1 x2 y2) representing the lower-left and upper-right corners of the search area.
`?layers`	List of layer names on which the `drawing` purpose metal shapes that are overlapped by `slot` purpose shapes should be converted to slotted metal shapes. The default is all layers.

Value Returned

`t`	The command completed. If `drawing` purpose shapes were found with overlapping `slot` purpose shapes, the `drawing` purpose shapes were converted to slotted `drawing` purpose shapes and the original shapes were removed.
`nil`	The conversion failed.

Example

```
lxConvertSlotToPolygon(?all)
```

Converts the `drawing` purpose metal shapes on all the layers that are overlapped by `slot` purpose shapes to slotted `drawing` purpose shapes.

```
lxConvertSlotToPolygon(?region list(0 0 1 1} ?layers list("Metal2"))
```

Converts the `Metal2 drawing` purpose metal shapes that are overlapped by `Metal2 slot` purpose shapes within the area given by the lower-left coordinate of (0,0) and the upper-right coordinate of (1,1) to slotted `Metal2 drawing` purpose shapes.

lxCreateBndFile

```
lxCreateBndFile(
     [ t_ixfFile ]
     [ t_netFile ]
     [ t_bndFile ]
     [ d_layId ]
     [ g_graphical ]
     )
     => t / nil
```

Description

Creates a binding file based on the PVS instance cross-reference file and the extracted netlist file.

Arguments

t_ixfFile	Path to a PVS LVS .ixf instance cross-reference file. **Note:** PVS creates a .ixf cross-reference file in the svdb directory of the PVS run directory when using the PVS **Create QRC Input Data** option.
t_netFile	Path to a PVS LVS .net extracted netlist file. **Note:** PVS creates a .net extracted netlist file in the svdb directory of the PVS run directory when using the PVS **Create QRC Input Data** option.
t_bndFile	Name of the binding file to be created.
d_layId	Database ID of the layout cellview corresponding to the PVS instance cross-reference file and the extracted netlist file.
g_graphical	Enables display of warning messages to indicate issues, if any, with the binding file creation.

Value Returned

`t`	The binding file was successfully created.
`nil`	Binding file creation failed.

Example

```
lxCreateBndFile ("./inv.ixf" "./inv.net" "./inv.bnd" layCV)
```

lxCreateGroupArray

```
lxCreateGroupArray(
     d_cvId
     l_figList
     [ ?name t_name ]
     [ ?rows x_rows ]
     [ ?columns x_cols ]
     [ ?X x_xValue ]
     [ ?Y x_yValue ]
     [ ?orients l_orientList ]
     [ ?mode 'pitch | 'spacing ]
     [ ?lpp l_refLPP ]
     )
     => d_groupArrayId / nil
```

Description

Creates a group array with the specified parameters.

Arguments

d_cvId	Database ID of the cellview in which the group array is to be created.
l_figList	List of figures for the first cell of the group array. Copies of the figures are repeated depending on the rows and columns specified for the group array.
?name *t_name*	Name of the group array.
?rows *x_rows*	Number of rows in the group array.
?columns *x_cols*	Number of columns in the group array.
?X *x_xValue*	Spacing added between cells of the group array in the X direction. Spacing is applied based on the mode specified.
?Y *x_yValue*	Spacing added between cells of the group array in the Y direction. Spacing is applied based on the mode specified.
?orients *l_orientList*	Orientation pattern used for the group array. If the pattern is smaller than the given rows and columns in the group array, the pattern is repeated from left to right and bottom to top. The bottom-most row in the pattern is considered as the first row.

`?mode 'pitch | 'spacing`

Mode used for applying spacing between group array cells. Valid values are `'pitch` and `'spacing`.

- `'pitch`: Lets you specify the distance that is measured between the same-edge corners of cells. For example, the distance between the left edges of two cells.
- `'spacing`: Lets you specify the distance that is measured between the adjacent or closest edges of cells. For example, the distance between the top edge of a cell and the lower edge of another cell.

`?lpp` *`l_refLPP`*

Specifies the layer-purpose pair used for computing the bounding box of the cells in a group array. The combined bounding box of the geometries on the specified reference layer-purpose pair is used for calculating the spacing between cells.

Value Returned

`d_groupArrayId`	Database ID of the group array upon successful creation.
`nil`	The group array could not be created.

Example

The following code creates a rectangle in the current cellview and saves its database ID in `rect`. Then, the rectangle is used to create a `2x2` group array named `myGroupArray` with the specified parameters. After the group array is created, `0x2ab9941a` is returned as its database ID and is saved in `groupArray1`.

```
cv=geGetEditCellView()
rect = dbCreateRect(cv list("Poly" "drawing") list(list(-5 0) list(0 5)))
groupArray1=lxCreateGroupArray(cv list(rect) ?name "myGroupArray" ?rows 2 ?columns
2 ?X 2 ?Y 3 ?orients list("R0") ?mode 'spacing ?lpp list("Poly" "drawing"))
==> 0x2ab9941a
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxCreateSynchronousClonesFromFigGroups

```
lxCreateSynchronousClonesFromFigGroups(
     ( figGroupList )
     [ dumpConsistencyCheckMsg ]
     [ dumpNonClonableFigGroups ]
     )
     => t / nil
```

Description

Creates a new clone family from a list of basic figGroups. When the clone family is created, the function runs the Clone Consistency Checker to verify that the figGroups are effectively synchronous. The checks that are run include verifying that the figGroups are identical, which means their shapes exist on the same layer, instances belong to the same master, and have the same relative position, and so on.

See Additional Information.

Arguments

figGroupList List of figGroups to be transformed to synchronous clones.

dumpConsistencyCheckMsg

Displays the messages issued during the Clone Consistency Check.

The default is `nil`.

dumpNonClonableFigGroups

Displays the list of figGroups, such as the `placement area` figGroup, that cannot be cloned or that contain figures that cannot be cloned.

Value Returned

`t`	The specified figGroups in the figGroup list are grouped in the syncClone figGroup to create a new clone family.
`nil`	Error with `syncClone` figGroup creation, synchronous clone family not created.

Additional Information

During the Clone Consistency Check, the checker may do one of the following:

1) Keep all the figGroups in the clone family when they are all found to be identical. In this case, all the figGroups have their type changed to "`syncClone`".

2) Keep part of the figGroups in the clone family. Identical figGroups get their type changed to "`syncClone`", other figGroups are removed form the clone family and kept as basic figGroups of type "`none`".

3) None of the figGroups are retained because all the figGroups are found to be different. In this case, the clone family is destroyed and all the figGroups are kept as basic figGroups of type "`none`".

4) Identify subgroup of the identical figGroups. In this case, the original clone family is split into several clone families. Each family gathers a set of identical figGroups. Each figGroup belonging to a clone family gets their type changed to "`syncClone`". Other figGroups that are not identical to any other figGroup are kept as basic figGroups of type "`none`".

Example

```
figGroupList = deGetCellView()~>figGroups
(db:0x29687b9a db:0x29687b9b db:0x29687b9c db:0x29687b9d db:0x29687b9e
db:0x29687b9f db:0x29687ba0
)
>
figGroupList~>type
("none" "none" "none" "none" "none"
"none" "none"
)
>
lxCreateSynchronousClonesFromFigGroups(figGroupList)
t
>
figGroupList~>type
```

```
("syncClone" "syncClone" "syncClone" "syncClone" "syncClone"
"syncClone" "syncClone"
)
>
=> Gathers all the identical figGroups of the opened cellView into clone family.

Select few figGroups in the layout cellView, then
figGroupList = geGetSelSet()
(db:0x29687b9b db:0x29687b9c db:0x29687b9a)
>
figGroupList~>objType
("figGroup" "figGroup" "figGroup")
>
figGroupList~>type
("none" "none" "none")
>
lxCreateSynchronousClonesFromFigGroups(figGroupList t)
INFO (LX-1350): The synchronous clone consistency check is running.
INFO (LX-1368): The inst (db:0x2968de1c) in clone "cloneFamily0_0" and the inst
(db:0x2968de1e) in clone "cloneFamily0_1" are different.
INFO (LX-1362): Clone 'cloneFamily0_1' has been removed from family 'cloneFamily0'
by cloning consistency check.
>
hiDBoxOK(lxCCCWarnPanel)
t
t
>
figGroupList~>type
("none" "syncClone" "syncClone")
>
```

lxDeleteSchematicDummies

```
lxDeleteSchematicDummies(
     [ ?schCV d_schCellViewID ]
     [ ?layCV d_layCellViewID ]
     [ ?layInsts l_insts ]
     [ ?useMFactor { t | nil } ]
     )
     => t / nil
```

Description

Deletes bound back annotated schematic instances for the specified dummy layout instances. For mfactored layout devices that have all the layout dummies included in the layout selected set, the SKILL function deletes the schematic dummy instances corresponding to the layout dummies. If all the layout dummy instances of the mfactored device are not included in the selected set, the SKILL function decrements the mfactor to reflect the removal of the selected dummies.

Arguments

`?schCV`	Database ID of the schematic cellview. If not specified, the current schematic cellview is used.
`?layCV`	Database ID of the layout cellview. If not specified, the current layout cellview is used.
`?layInsts`	List of layout instances for which the bound back annotated schematic dummy instances need to be deleted. If not specified, all the specified layout instances are considered for bound schematic deletion.
`?useMFactor`	Deletes the schematic dummy instances corresponding to the layout dummies of an mfactored device if all the layout dummies of the mfactor are included in the selected set of layout instance. The default is `t`. When set to `nil`, the schematic dummy instances bound to the layout dummies of an mfactored device is deleted even when all the layout dummies are not included in the selected set.

Value Returned

`t`	Back annotated schematic dummy instances bound to the specified layout instances are deleted.
`nil`	Bound schematic dummy instances are not deleted.

Example

```
lxDeleteSchematicDummies(?layInsts geGetSelectedSet() ?useMFactor t)
```

Decrements the schematic dummies bound to dummies in the layout selected set of instances in the current layout cellview. If all the layout dummies of an mfactor are selected, the schematic dummy is deleted.

lxDeleteSynchronousCloneFamily

```
lxDeleteSynchronousCloneFamily(
     [ cloneList ]
     )
     => t / nil
```

Description

Deletes the specified family of synchronous clones. As a result, the member clones are desynchronized and they behave as basic groups.

Argument

cloneList The cloneList can be:

- A single synchronous clone
- A list of synchronous clones
- Nothing specified

Note:

- When the function is run with no argument specified, the function is called on selected figures, such as, when calling `lxDeleteSynchronousCloneFamily(geGetSelSet())`
- If the argument list contains a non-synchronous clone object, the object is ignored.

Value Returned

`t`	At least one synchronous clone family has been deleted.
`nil`	No clone families deleted.

Example

```
lxDeleteSynchronousCloneFamily (list (clone1 clone2))
```

lxFold

```
lxFold(
     l_insts
     l_point
     x_numFolds
     [ ?widths l_widths ]
     [ ?chainFolds { t | nil } ]
     [ ?ignoreMFactor { t | nil } ]
     [ ?chainAlignPMOS { Top | Center | Bottom } ]
     [ ?chainAlignNMOS { Top | Center | Bottom } ]
     [ ?enforceWidthMatch { t | nil } ]
     [ ?keepFolds { t | nil } ]
     )
     => l_chains / nil
```

Description

Folds a list of valid instances into the specified number of folds and places them at the specified location. If the disableFolding environment variable is set and the folding threshold for the component type is set to `0`, folding is disabled.

Arguments

l_insts	List of instances to be folded.
l_point	Position of the folds generated in the layout.
x_numFolds	The number of folds that each instance will be folded into. Can be any integer value. **Note:** If folding on multi-fingered devices, this argument is ignored. For more details, refer to the foldFingerSplitUseSchWidth environment variable and Folding a Multi-fingered Device to Create Split Fingers.
?chainFolds	Abuts the folds of newly folded devices into chains. The default is `t`. Any existing folds are deleted, except the one that is passed to the SKILL function, which is then used to recreate the new folds.

`?widths` *`l_widths`*

List of widths specifying the width of each fold. This value is required if you do not want each fold to have the same width.

Note: If folding on multi-fingered devices, this argument is ignored. Instead, the width is determined by the setting specified for foldFingerSplitUseSchWidth environment variable. For more details, refer to Folding a Multi-fingered Device to Create Split Fingers.

`?ignoreMFactor`

Ignores the schematic multiplication factor when generating folded devices.

The default is `nil`.

`?chainAlignPMOS`

Controls PMOS chaining alignment using values: `"Top"`, `"Center"`, and `"Bottom"`. The values should be enclosed in quotation marks.

The default is `"Top"`.

`?chainAlignNMOS`

Controls NMOS chaining alignment using values: `"Top"`, `"Center"`, and `"Bottom"`. The values should be enclosed in quotation marks.

The default is `"Bottom"`.

`?enforceWidthMatch`

Enforces that the device folds are generated with the same total width as that of the corresponding schematic device.

The default is `nil`, which triggers a message warning about the total fold width being different from that of the schematic but allows the folds to be generated.

`?keepFolds`

Folds the existing folds.

The default is set by the `foldKeepFolds` environment variable.

Value Returned

l_chains	List of folds generated.
nil	The specified instances were not generated as folded devices.

Tip

All arguments that take a string value must have their values enclosed in quotation marks.

Examples

Example 1

```
lxFold(list(P0) 0:0 3)
```

Creates three folds, `P0.1`, `P0.2`, and `P0.3` positioned at `0:0`. The generated folds will be chained if the environment variable, `chainFolds`, is set to `true`.

Returns `((dbId dbId dbId))` where each `dbId` is one of the three folds created.

Example 2

```
lxFold(list(P0 N0) 10:10 2 ?chainFolds nil)
```

Creates four folds, `P0.1`, `P0.2 N0.1`, and `N0.2` positioned at `10:10`, which will not be folded.

Returns `((dbId dbId) (dbId dbId))` where the first list is `P0.1` and `P0.2` and the second list is `N0.1` and `N0.2`.

lxGenerateFinish

```
lxGenerateFinish(
     d_schId
     d_layId
     [ ?extractSchematic { t | nil } ]
     )
     => t / nil
```

Description

Finishes the layout generation process by executing the *Generate All From Source* command with the form values supplied by `lxSet*` functions and after the previous call to `lxGenerateStart`.

When used with the other `lxSet*` functions, this provides an alternative interface to the Generate Layout form. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

d_schId	Database ID of the schematic cellview that was used as the connectivity source.
d_layId	Database ID of the layout cellview in which components were generated.
?extractSchematic	Automatically extracts the schematic if required. If the schematic needs to be extracted and this option is not set, the system issues a message and layout generation stops.

Value Returned

t	Layout generation completed.
nil	Layout generation did not complete. Check the log file for details.

Example

```
lxGenerateStart(schCv layCv (?extractSchematic t))
...
;; lxSet* calls used to set form values
```

```
....
lxGenerateFinish(schCv layCv (?extractSchematic t))
```

lxGenerateGetAvailableBoundaryLPPs

This function is no longer supported and will be ignored if specified. It has been retained to maintain backward compatibility.

lxGenerateGetPinNets

This function is no longer supported. Use lxGetPinNets instead.

lxGenerateSetAreaEstimationOptions

This function is no longer supported. Use lxSetAreaEstimationOptions instead.

lxGenerateSetBoundaryOptions

This function is no longer supported. Use lxSetBoundaryOptions instead.

lxGenerateSetGenerateOptions

This function is no longer supported. Use lxSetGenerateOptions instead.

lxGenerateSetNetPinSpecs

This function is no longer supported. Use lxSetNetPinSpecs instead.

lxGenerateStart

```
lxGenerateStart(
     d_schId
     d_layId
     [ ?extractSchematic { t | nil } ]
     )
     => t / nil
```

Description

Starts the *Generate All From Source* command, building the Generate Layout form without displaying it to the screen.

When used with the other `lxSet*` functions, this provides an alternative interface to the Generate Layout form. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

d_schId	Database ID of the schematic cellview to be used as the connectivity source.
d_layId	Database ID of the layout cellview in which components will be generated.
?extractSchematic	Automatically extracts the schematic if required. If the schematic needs to be extracted and this option is not set, the system issues a message and layout generation stops.

Value Returned

t	The Generate Layout form was initialized.
nil	The Generate Layout form was not initialized. Check that you have a Layout XL license available, and then check the log file for details.

Example

```
hiSetCurrentWindow(schWin)
schCv = geGetEditCellView()

hiSetCurrentWindow(layWin)
```

```
layCv = geGetEditCellView()
....

 lxGenerateStart(schCv layCv (?extractSchematic t))
```

Opens the Generate Layout form for the current views in the schematic and layout windows and extracts the schematic if required.

lxGenFromSource

```
lxGenFromSource(
     d_schCellViewID
     [ ?layLibName t_layLibName ]
     [ ?layCellName t_layCellName ]
     [ ?layViewName t_layViewName ]
     [ ?configLib t_configLib ]
     [ ?configCell t_configCell ]
     [ ?configView t_configView ]
     [ ?initCreateInstances { t | nil } ]
     [ ?initDoStacking { t | nil } ]
     [ ?initDoFolding { t | nil } ]
     [ ?initCreatePins { t | nil } ]
     [ ?initGlobalNetPins { t | nil } ]
     [ ?initCreatePadPins { t | nil } ]
     [ ?initCreateBoundary { t | nil } ]
     [ ?initCreateSnapBoundary { t | nil } ]
     [ ?lxPositionMinSep n_lxPositionMinSep ]
     [ ?lxGenerateInBoundary { t | nil } ]
     [ ?initCreateMTM { t | nil } ]
     [ ?extractAfterGenerateAll { t | nil } ]
     [ ?extractSchematic { t | nil } ]
     [ ?ignoreSchematicCheck { t | nil } ]
     [ ?initPrBoundaryShape t_initPrBoundaryShape ]
     [ ?initPrBoundaryOrigin l_initPrBoundaryOrigin ]
     [ ?initPrBoundaryPoints l_initPrBoundaryPoints ]
     [ ?initAreaSource1 t_initAreaSource1 ]
     [ ?areaCalc t_areaCalc ]
     [ ?initAreaSource2 t_initAreaSource2 ]
     [ ?initAreaSource1Val n_initAreaSource1Val ]
     [ ?initAreaSource2Val n_initAreaSource2Val ]
     [ ?virtualHierarchy { t | nil } ]
     [ ?softBlocks { t | nil } ]
     [ ?softBlockArea { t | nil } ]
     [ ?useAreaBoundaryUtilization { t | nil } ]
     [ ?areaBoundaryEnclosure n_areaBoundaryEnclosure ]
     [ ?allAreaBoundaries { t | nil } ]
     )
     => dbCellViewID
```

Description

Generates a layout from a specified schematic cellview. If you do not specify the layout library, cell, and view names, the layout view is created with the same name as the schematic cell. If you do not specify the physConfig library, cell and view names, the layout view is created with the same name as the layout cell with `physConfig` as the default physConfigView. If

you use the optional arguments, they override the existing environment variables set in the `.cdsenv` file.

Arguments

d_schCellViewID

Database ID of the source schematic cellview.

?layLibName "*layLibName*"

Name of the library in which the layout is to be generated.

?layCellName "*layCellName*"

Name of the layout cell to be generated.

?layViewName "*layViewName*"

Name of the layout view to be generated.

?configLib *t_configLib*

Name of the library in which the physical configuration is stored.

?configCell *t_configCell*

Name of the cell in which the physical configuration is stored.

?configView *t_configView*

Name of the view in which the physical configuration is stored.

?initCreateInstances

Generates all the instances in the schematic that do not have one of the ignore properties attached to them.

?initDoStacking

Automatically abuts MOS transistors into chains during layout generation.

?initDoFolding

Automatically splits devices into fingers to prevent gate width from exceeding a specific size.

?initCreatePins

Generates the pins in the design and snaps them to the placement grid.

`?initGlobalNetPins`

Generates layout pins for the global nets in the schematic.

This argument is honored only if `?initCreatePins` is set to `t`.

`?initCreatePadPins`

Generates layout pins and pads for schematic pins that are connected to I/O pads.

This argument is honored only if `?initCreatePins` is set to `t`.

`?initCreateBoundary`

Generates a place and route boundary based on the settings you make in the *Boundary* tab of the Generate Layout form.

`?initCreateSnapBoundary`

Generates a rectangular snap boundary that encloses the generated place and route boundary.

This argument is honored only if `initCreateBoundary` is also `t`.

`?lxPositionMinSep` *`n_lxPositionMinSep`*

Positions the instances in the layout at a minimum separation based on the user-defined value.

`?lxGenerateInBoundary`

Generates layout representations within the design boundary when the Generate All From Source command is run.

`?initCreateMTM`

Preserves user-defined one-to-one, many-to-many, many-to-one, and one-to-many device correspondence defined in the Define Device Correspondence form. It does not preserve user-specified internal bindings made using the Choose Binding form; nor does it report missing devices or shapes within a bound group.

`?extractAfterGenerateAll`

Automatically extracts the connectivity of the layout design after layout generation is complete.

`?extractSchematic`

Automatically extracts the schematic design if required.

If the schematic needs to be extracted and this option is not set, the system issues a message and layout generation stops.

`?ignoreSchematicCheck`

Overrides the value of the `?extractSchematic` option.

Use this option to restore the default behavior of releases prior to IC 6.1.4, whereby if schematic extraction is required, `lxGenFromSource` issues a warning and asks you to rerun the function, but generates the layout in any case.

`?initPrBoundaryShape` *`t_initPrBoundaryShape`*

Specifies the shape of the prBoundary. The valid shapes are `Rectangle` and `Polygon`.

The default shape of the prBoundary is `Rectangle`.

`?initPrBoundaryOrigin` *`l_initPrBoundaryOrigin`*

Specifies the co-ordinates of the prBoundary's origin.

This argument is required when `initPrBoundaryShape` is set to `rectangle`.

`?initPrBoundaryPoints` *`l_initPrBoundaryPoints`*

Specifies the set of points that define the vertices of the prBoundary when the selected boundary shape is polygon.

This is a required argument when `initPrBoundaryShape` is set to `polygon`.

`?initAreaSource1` *`t_initAreaSource1`*

Specifies the first parameter that defines how the area of the prBoundary must be calculated.

The valid values for this argument are:

- *Utilization*
- *AspectRatio*
- *BoundaryWidth*
- *BoundaryHeight*

The default value is `Utilization`.

`?areaCalc` *`t_areaCalc`* Specifies the area calculation function to use.

The valid values are *PR Boundary* and *BBox*.

The default is `PR Boundary`.

`?initAreaSource2` *`t_initAreaSource2`*

Specifies the second parameter that defines how the area of the prBoundary must be calculated. The value of this argument must be different from the value set for `initAreaSource1`.

The valid values for this argument are:

- *Utilization*
- *AspectRatio*
- *BoundaryWidth*
- *BoundaryHeight*

The default value is `AspectRatio`.

`?initAreaSource1Val` *`n_initAreaSource1Val`*

Specifies the value of the parameter specified by `initAreaSource1`, which will be used to calculate the area of the PR Boundary.

The default value is `0.25`, which is determined using the environment variables `initUtilization`, `initAspectRatio`, `initPRBoundaryW` and `initPRBoundaryH`.

Note: The value you specify using this argument is considered to be equivalent to `<value>%/100`. Therefore, a value of `100` specified using the argument will result in an effective area utilization value of:

```
100%/100 = 1
```

`?initAreaSource2Val` *`n_initAreaSource2Val`*

Specifies the value for the parameter specified by `initAreaSource2`, which will be used to calculate the area of the PR Boundary.

The default value is `1`, which is determined using the environment variables `initUtilization`, `initAspectRatio`, `initPRBoundaryW` and `initPRBoundaryH`.

`?virtualHierarchy`

Generates virtual hierarchy for schematic symbols that do not have any corresponding layouts.

The default value is `nil`.

`?softBlocks`

Generates soft blocks for schematic symbols that have a missing schematic.

The default value is `t`, but for the soft block to be created the `?virtualHierarchy` argument should also be set to `t`.

`?softBlockArea` *`n_softBlockArea`*

Specifies the area of the soft block.

The default is `100.0`.

`?useAreaBoundaryUtilization`

Controls whether the areaBoundaryUtilization environment variable is used to determine the area boundary size for the virtual hierarchy.

The default is `nil`, which means the areaBoundaryEnclosure environment variable is used to calculate the size of the area boundary.

`?areaBoundaryEnclosure` *`n_areaBoundaryEnclosure`*

Specifies the floating point value that defines the the minimum distance from the area boundary edge to the instances inside.

`?allAreaBoundaries`

Controls whether the specified area boundary settings are used for generating the virtual hierarchies at all levels in the hierarchy or only for the top-level virtual hierarchy.

The default is `nil`, which means the specified area boundary settings are used only for generating the virtual hierarchy area boundaries for the top-level virtual hierarchy.

Value Returned

`dbCellViewID`	Database ID of the generated layout cellview.

Examples

Generates a layout for the schematic cellview, `schCV`, in the layout view, `layoutNew`. `layoutNew` is the argument specified by *`layViewName`* in the same library cell as the schematic, using the default environment variable settings from the `.cdsenv`:

```
schCV=dbOpenCellViewByType( schLibName schCellName schViewName )
lxGenFromSource( schCV ?layViewName "layoutNew")
```

You must perform an explicit `dbSave` after you have generated the new layout cellview.

Generates a layout for the schematic cellview, `schCV`, using a predefined physConfig (`configLib, configCell, configView`).

```
schCV=dbOpenCellViewByType( schLibName schCellName schViewName )
lxGenFromSource( schCV ?configLib "configLib" ?configCell "configCell"
?configView "configView")
```

By default `configLib` and `configCell` are same as schematic cellview and configView is `physConfig`.

Generates a virtual hierarchy for the missing schematic symbols in the schematic cellview, `scv`.

```
lxGenFromSource(scv
?layLibName lcv~>libName
?layCellName lcv~>cellName
?layViewName lcv~>viewName
?initDoFolding t
?lxGenerateInBoundary t
?initAreaSource1 "Utilization"
?initAreaSource1Val 0.15
?initAreaSource2 "AspectRatio"
?initAreaSource2Val 1.0
?virtualHierarchy t
?softBlocks nil
?softBlockArea 200
?areaBoundaryEnclosure 1.0
?allAreaBoundaries t)
```

lxGetAvailablePinLPPs

```
lxGetAvailablePinLPPs(
     )
     => lt_layerPurposePair / nil
```

Description

Lists the valid layer-purpose pairs to use for setting the pin layer. This function must always be called between calls to `lxGenerateStart` and `lxGenerateFinish` or `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`. It is typically used to find a layer-purpose pair to pass to `lxSetNetPinSpecs`.

Arguments

None

Value Returned

`lt_layerPurposePair`
: List of layer-purpose pairs.

`nil`
: The function failed.

Examples

```
lxGenerateStart(schCv layCv (?extractSchematic t))
...
  pinLPPs = lxGetAvailablePinLPPs()
...
lxGenerateFinish(schCv layCv (?extractSchematic t))
```

Gets a list of valid layer-purpose pairs from the Generate Layout form.

```
lxUpdateComponentsAndNetsStart(schCv layCv (?extractSchematic t))
...
  pinLPPs = lxGetAvailablePinLPPs()
...
lxUpdateComponentsAndNetsFinish(schCv layCv (?extractSchematic t))
```

Gets a list of valid layer-purpose pairs from the Update and Components and Nets form.

lxGetCloneFamilyName

```
lxGetCloneFamilyName(
     clone
     )
     => familyName / nil
```

Description

Returns the family name of the specified clone.

Arguments

clone	The clone of which the family name needs to be determined.

Value Returned

familyName	Name of the specified clone's family.
nil	The specified object is not a synchronous clone.

Example

```
lxGetCloneFamilyName(clone)
```

lxGetConnRef

```
lxGetConnRef(
     { w_windowID | d_layCellviewID }
     => l_connref
```

Description

Gets the connectivity reference for the specified layout cellview or for the layout cellview opened in the specified window. The layout cellview does not need to be open in a Layout XL session for the connectivity reference to be returned.

Arguments

w_windowID

Window ID of the layout cellview for which the connectivity reference needs to be returned.

d_layCellviewID

Layout cellview ID for which the connectivity reference needs to be returned.

Value Returned

l_connRef

A list that displays the connectivity reference coordinates for the specified window or layout cellview ID.

Examples of values returned:

- `("NONE" "" "" "" "")`

where, `"NONE"` implies that no connectivity reference is defined for the layout cellview. If Layout XL is launched for this cellview, it will open without a schematic.

- `("CELLVIEW" "<libName>" "<cellName>" "<viewName>" "")`

where, the schematic library, cell, and view name are displayed. The last entry in the list `""` can be ignored.

Example

```
connRef = lxGetConnRef(hiGetCurrentWindow())
connRef = lxGetConnRef(geGetEditCellView())
```

lxGetEditedSyncClone

```
lxGetEditedSyncClone(
     d_layCellViewId
     )
     => d_syncCloneId / nil
```

Description

Returns the database ID of the synchronous clone currently being edited in the specified layout cellview. If the function finds a currently-edited layout group which is not a synchronous clone, it recursively checks to determine if its parent group is a synchronous clone. If it is, the function returns the ID of the parent synchronous clone.

Arguments

d_layCellViewId	Database ID of the layout cellview.

Value Returned

d_syncCloneId	Database ID of the synchronous clone currently being edited.
nil	There is no synchronous currently being edited.

Example

```
lxGetEditedSyncClone(layCv)
```

lxGetGroupArrayParams

```
lxGetGroupArrayParams(
     d_groupArrayId
     )
     => l_params / nil
```

Description

Returns parameters of the specified group array.

Arguments

d_groupArrayId	Database ID of the group array whose parameters you want to view.

Value Returned

l_params	List of parameters if the group array is valid.
nil	The group array is invalid.

Example

The following code returns the parameters of the `myGroupArray` group array whose database ID is stored in `groupArray1`.

```
lxGetGroupArrayParams(groupArray1)
==> (nil mode "spacing" Y 3.0
    X 2.0 column 2 row
    2 name "myGroupArray" orients "(\"R0\")"
    lpp "Poly drawing"
    )
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxGetLXInfo

This function is obsolete, use lxGetSource instead.

lxGetOtherClonesInFamily

```
lxGetOtherClonesInFamily(
     clone
     )
     => cloneList / nil
```

Description

Returns a list of clones in a family other than the clone specified.

Note: The generated list does not contain the specified clone.

Arguments

`clone`	The clone for which the other family members are listed.

Value Returned

`cloneList`	List of all the clones in the family except the clone itself.
`nil`	The specified object is not a synchronous clone.

Examples

```
lxGetOtherClonesInFamily (clone)
```

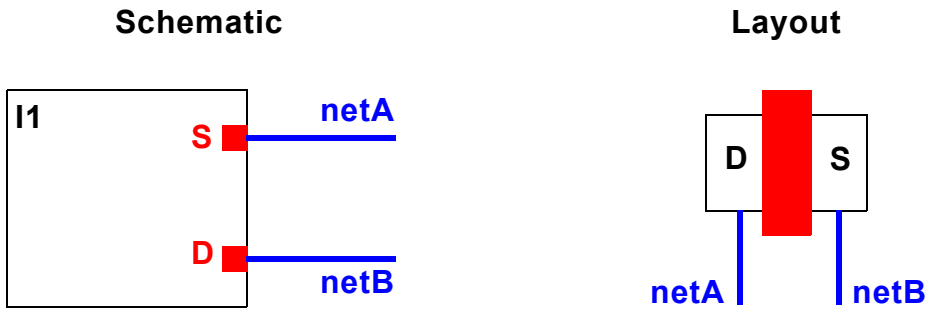
lxGetPermutedInstTerms

```
lxGetPermutedInstTerms(
     d_layInstID
     )
     => ld_instTermIDs / nil
```

Description

Returns the list of database IDs of instance terminals that have been permuted on a specified layout instance. The function can identify instance terminals that have been permuted, even if there is no permuteRule property on the schematic or layout instances or their masters.

In the example below, the connectivity of instance terminals `S` and `D` has been permuted in the layout implementation of schematic instance `I1`. `lxGetPermutedInstTerms` would return the database IDs for instance terminals `D` and `S` in the layout instance.



Arguments

d_layInstID	Database ID of a layout instance.

Value Returned

ld_instTermIDs	List of database IDs for instance terminals that have been permuted.
`nil`	The layout instance has no permuted instance terminals.

Example

```
lxGetPermutedInstTerms(instID)
```

lxGetPinNets

```
lxGetPinNets(
     )
     => lt_netName / nil
```

Description

Lists the pin net names from the Generate Layout or Update Components and Nets form. This function must always be called between calls to `lxGenerateStart` and `lxGenerateFinish` or `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`. It is typically used to find net names to pass to `lxSetNetPinSpecs`.

Arguments

None

Value Returned

lt_netName	List of net names.
nil	The function failed.

Examples

```
lxGenerateStart(schCv layCv (?extractSchematic t))
...
  nets = lxGetPinNets()
...
lxGenerateFinish(schCv layCv (?extractSchematic t))
```

Gets the pin names from the Generate Layout form.

```
lxUpdateComponentsAndNetsStart(schCv layCv (?extractSchematic t))
...
  nets = lxGetPinNets()
...
lxUpdateComponentsAndNetsFinish(schCv layCv (?extractSchematic t))
```

Gets the pin names from the Update Components and Nets form.

lxGetSchematic

```
lxGetSchematic(
     d_layoutCellViewId
     )
     => d_schematicCellViewId / nil
```

Description

Returns either the schematic currently open in an existing Layout XL session, the schematic defined by the connectivity reference returned by `lxGetConnRef`, or the schematic in the same library and cell as the layout when no connectivity reference is defined.

Argument

d_layoutCellViewId

Database ID of a layout cellview

Value Returned

d_schematicCellViewId

Database ID of a schematic cellview.

`nil` No schematic was found or an error occurred.

Example

Retrieves the ID for the Layout XL cellview being edited currently and then returns the database ID of the schematic cellview open for this Layout XL cellview.

```
lxGetSchematic(geGetEditCellView())
> db:0x3594001a
```

Related Topics

lxGetConnRef

lxGetSource

```
lxGetSource(
     { w_windowID | d_layCellviewID }
     => d_schCellviewID / nil
```

Description

Returns the source (schematic) cellview ID for the specified layout window ID or layout cellview ID in the current Layout XL session. If the schematic reference for the specified layout window has changed since the current Layout XL session was started, Layout XL must be relaunched for `lxGetSource` to return the updated schematic reference.

Arguments

w_windowID

Layout window ID corresponding to the layout cellview ID for which the schematic cellview ID needs to be returned.

d_layCellviewID

Layout cellview ID for which the schematic cellview ID needs to be returned.

Value Returned

d_schCellviewID	Schematic cellview ID corresponding to the layout cellview ID open in the current Layout XL session.
`nil`	The function failed.

Examples

```
source = lxGetSource(hiGetCurrentWindow())
```

or

```
source = lxGetSource(geGetEditCellView())
```

lxGetSyncClone

```
lxGetSyncClone(
     d_objectId
     )
     => d_syncCloneId / nil
```

Description

Returns the database ID of the synchronous clone to which the specified object belongs. If the object belongs to a layout group which is not a synchronous clone, the function recursively checks to determine if its parent group is a synchronous clone. If it is, the function returns the ID of the parent synchronous clone.

Arguments

d_objectId	Database ID of an object in the current cellview.

Value Returned

d_syncCloneId	Database ID of the synchronous clone to which the specified object belongs.
nil	The object does not belong to a synchronous clone.

Example

```
lxGetSyncClone(objId)
```

lxGetValidViaDefs

```
lxGetValidViaDefs(
     d_objID
     [ ds_constraintGroup ]
     [ ltx_bottomLayer ]
     [ ltx_topLayer ]
     )
     => list_of_viaDefs / nil
```

Description

Returns a list of via definitions associated with a specified design object. You can restrict the list to include only vias defined in a specified constraint group or which feature a specific top and/or bottom layer. If you specify a technology file, no constraint groups, and no layers, the function returns all the via definitions in the technology database. If you specify both top and bottom layers, the function returns only via definitions featuring those layers. If you specify either a top or bottom layer, the function returns only via definitions featuring the layer in question.

Arguments

d_objID

Database ID of the object to be queried for associated via definitions.

Valid object types are: net, route, cellview, or technology file.

ds_constraintGroup

Name or database ID of the constraint group in which the via is defined.

ltx_bottomLayer

Bottom layer of the via definition.

ltx_topLayer

Top layer of the via definition.

Value Returned

list_of_viaDefs

List of via definitions associated with the specified design object or featuring the specified top and/or bottom layer.

- If you do not specify a top or bottom layer, the function returns a list of all the valid via definitions.
- If you specify top *and* bottom layers, the function returns a list of lists, each list containing the valid vias between pairs of layers from the specified layer range.

See the examples below for more information.

`nil`

The command was unsuccessful.

Examples

Returns a list of valid vias defined between layers `Metal3` and `Metal4`.

```
lxGetValidViaDefs(geGetEditCellView() nil "Metal3" "Metal4")
((db:0x149dd397 db:0x149dd3af db:0x149dd3a6))
```

Returns two lists of vias, one for those defined between layers `Metal2` and `Metal3` and the other for vias defined between layers `Metal3` and `Metal4`.

```
lxGetValidViaDefs(geGetEditCellView() nil "Metal2" "Metal4")
```

```
((db:0x149dd397 db:0x149dd3af db:0x149dd3a6)
    (db:0x149dd3b5 db:0x149dd398 db:0x149dd3b0 db:0x149dd3a7)
)
```

Returns a single list of all the valid vias defined for the current cellview.

```
lxGetValidViaDefs(geGetEditCellView() nil)
(db:0x149dd39c db:0x149dd39b db:0x149dd39f db:0x149dd39e db:0x149dd39d
    db:0x149dd3b2 db:0x149dd3a0 db:0x149dd3a9 db:0x149dd39a db:0x149dd3b3
    db:0x149dd3a8 db:0x149dd3b1 db:0x149dd399 db:0x149dd812 db:0x149dd813
    db:0x149dd3b4 db:0x149dd3a7 db:0x149dd3b0 db:0x149dd398 db:0x149dd3b5
    db:0x149dd3a6 db:0x149dd3af db:0x149dd397 db:0x149dd3a5 db:0x149dd3ae
    db:0x149dd396 db:0x149dd3a4 db:0x149dd3ad db:0x149dd395 db:0x149dd3a3
    db:0x149dd3ac db:0x149dd394 db:0x149dd3a2 db:0x149dd3ab db:0x149dd393
    db:0x149dd3a1 db:0x149dd3aa db:0x149dd392
)
```

Creates a via using the lxGetValidViaDefs, lxComputeViaParams, and `dbCreateVia` commands.

1. `cellViewId = geGetEditCellView()`
2. `viaDefs=lxGetValidViaDefs(cellViewId "highYield")`
   `; pick a viaDefId`
3. `viaId=nth(3 viaDefs)`
4. `viaParams=lxComputeViaParams(viaId cellViewId "highYield"`
   `list(list("cutRows" 3) list("cutColumns" 3)))`
5. `viaId=dbCreateVia(cellViewId viaId list(5 5) "R0" viaParams)`

lxGroupArrayCreatePreset

```
lxGroupArrayCreatePreset(
     t_presetName
     t_orientPattern
     t_presetDesc
     g_isEditable
     )
     => t / nil
```

Description

Creates an orientation pattern for group arrays.

Arguments

t_presetName	Name of the orientation pattern.
t_orientPattern	Orientation pattern values in escaped string format.
t_presetDesc	Description of the orientation pattern.
g_isEditable	Boolean value to indicate whether the pattern can be modified and deleted.

Value Returned

t	The group array orientation pattern was created successfully.
nil	The operation failed.

Examples

Creates an orientation pattern named `myPattern`, with the orientation `MX` and `R0`. The pattern can be modified and deleted.

```
lxGroupArrayCreatePreset("myPattern" "\"MX\" \"R0\"" "Desc_my pattern" t)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxHiBackAnnotateDummies

```
lxHiBackAnnotateDummies(
     w_windowID
     )
     => t / nil
```

Description

Launches the interactive backannotation enter function to create dummy instances from the layout view into the corresponding schematic view.

Arguments

w_windowID	ID of the layout window in which to run the enter function. If the window ID is not specified, the current window is used.

Value Returned

`t`	Enter function ran successfully.
`nil`	Enter function did not start.

Examples

The enter function ran successfully in `window` and can be used to create dummies.

```
when(window = hiGetCurrentWindow()
  lxHiBackAnnotateDummies(?windowId window)
)
```

The enter function ran successfully in the current window and can be used to create dummies.

```
lxHiBackAnnotateDummies()
```

lxHierCheck

```
lxHierCheck(
     layCV
     [ ?checkDevices { t | nil } ]
     [ ?checkExtractLayout { t | nil } ]
     [ ?extractStopLevel [0-32] ]
     [ ?hierSummaryLogFileName t_hierSummaryLogFileName ]
     )
     => 0 - 100 / -1
```

Description

Checks all unique layout masters in the hierarchy for Layout XL compliance, and reports binding status.

Arguments

layCV	Database ID of the layout cellview.
?checkDevices	Checks the lower-level cellviews in the layout for opens and shorts when extracted to `extractStopLevel`.
?checkExtractLayout	Controls the extraction of layout nets for the layout hierarchy being checked for XL-compliance issues.
?extractStopLevel	Specifies the hierarchy depth for extraction when checking for opens and shorts. The default is `0`. The extraction depth must be an integer between `0` and `32`.
?hierSummaryLogFileName	Specifies the name of the file to which the Summary report of the hierarchical check is written.

Value Returned

`0 – 100`	The average compliance.
`-1`	The check failed.

Example

```
layCV=dbOpenCellViewByType(layLibName layCellName layViewName)
lxHierCheck(layCV ?checkDevices t ?extractStopLevel 2 ?hierSummaryLogFileName "./
hierCheck.log")
```

Checks all unique layout masters up to level 2 in the hierarchy of `layLibName`/`layCellName`/`layViewName`. The Summary report of the check is written to the log file named `hierCheck.log`.

Related Topics

Design Check for Layout XL Compliance

Binding Updates

lxHierCheckAgainstSource

```
lxHierCheckAgainstSource(
     d_layCellViewID
     [ ?masterDiff { t | nil } ]
     [ ?paramDiff { t | nil } ]
     [ ?unboundInsts { t | nil } ]
     [ ?connectivity { t | nil } ]
     [ ?maxDiffsLimit x_maxDiffsLimit ]
     [ ?logFileName t_logFileName ]
     [ ?appendToLog { t | nil } ]
     [ ?nameDiff { t | nil } ]
     [ ?dummyDiff { t | nil } ]
     )
     => t / nil
```

Description

Generates a report highlighting the differences between the schematic and layout cellviews for unique layout masters in the specified layout hierarchy. If you use the optional arguments, they override the existing environment variables set in the `.cdsenv` file. This SKILL function always skips any layouts that have no instances. If a log file name is specified for the hierSummaryLogFileName environment variable, the SKILL function prints the summary information to the specified log file, in addition to printing the information to the default Virtuoso log file.

Arguments

d_layCellViewID	Database ID of the target layout cellview.
?masterDiff	Reports instance masters that are missing or mismatched between the layout and the schematic.
?paramDiff	Reports CDF parameter mismatches and property mismatches between the schematic and layout views.
?unboundInsts	Reports layout instances that are not bound to schematic instances.
?connectivity	Reports connectivity mismatches on top level pins and global nets; mismatched or missing terminals and instance terminals; and unbound nets in the layout.

`?maxDiffsLimit` *`x_maxDiffsLimit`*

Limits the number of differences reported in the CIW for each of the four categories. For example, if you set the limit to 100, the command can report up to 100 cellview master differences and 100 unbound instance differences, and so on. Any messages over the limit are suppressed in the CIW but are listed in the log file.

`?logFileName` *`t_logFileName`*

Specifies the name of the file in which all the *Check Against Source* messages are written.

`?appendToLog`

Appends the results of the current run to the specified log file instead of overwriting the log file.

`?nameDiff`

Reports name differences for bound instances, if the Layout XL naming convention does not match that of the schematic. The name differences are updated by *Update Components And Nets*.

The default is `nil`.

Example:

If the `nameDiff` argument is set to `t`, and the schematic instance `INV/N0` is bound to the layout instance, `I0`, the layout instance name is updated to `INV|N0`. If the `prefixLayoutInstNamesWithPipe` environment variable is set to `t`, the layout instance name is prefixed with an OR (`|`) bar, as `|INV|N0`.

`?dummyDiff`

Reports layout dummy instances that may require backannotation to the schematic.

The default is `nil`.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Example

```
lxHierCheckAgainstSource(lcv)
```

Generates the hierarchical Design Check Against Source report for differences between the source and the target, using the default environment variable settings.

Note: `lcv` is the layout cellview ID. The schematic can be automatically determined from the connectivity reference setting on the layout. The connectivity reference can be changed, if required, using `lxSetConnRef()`.

```
lxHierCheckAgainstSource(lcv ?masterDiff t ?paramDiff nil ?unboundsInsts nil
?connectivity t ?logFileName "./checkReport.log")
```

Generates the *Check Against Source* report for differences between the source and the target. Only the differences in the masters and connectivity are reported. The log file is `checkReport.log`.

Note: `lcv` is the layout cellview ID.

lxHierUpdateComponentsAndNets

```
lxHierUpdateComponentsAndNets(
     d_layCellViewID
     [ ?initCreatePins { t | nil } ]
     [ ?initGlobalNetPins { t | nil } ]
     [ ?initCreatePadPins { t | nil } ]
     [ ?initCreateInstances { t | nil } ]
     [ ?initCreateBoundary { t | nil } ]
     [ ?initCreateSnapBoundary { t | nil } ]
     [ ?initDoStacking { t | nil } ]
     [ ?initDoFolding { t | nil } ]
     [ ?initCreateMTM { t | nil } ]
     [ ?deleteUnmatchedInsts { t | nil } ]
     [ ?deleteUnmatchedPins { t | nil } ]
     [ ?updateReplacesMasters { t | nil } ]
     [ ?updateWithMarkers { t | nil } ]
     [ ?updateLayoutParameters { t | nil } ]
     [ ?updateNetSigType { t | nil } ]
     [ ?updateNetMinMaxVoltage { t | nil } ]
     [ ?updateNetsOnly { t | nil } ]
     [ ?virtualHierarchy { t | nil } ]
     [ ?skipLeafs { t | nil } ]
     [ ?extractSchematic { t | nil } ]
     )
     => t / nil
```

Description

Updates the components and nets for all unique layout masters in the specified layout cellview hierarchy. The layouts are automatically opened in edit mode for update. If a log file name is specified for the hierSummaryLogFileName environment variable, the SKILL function prints the summary information to the specified log file, in addition to printing the information to the default Virtuoso log file.

Arguments

d_layCellViewID

Database ID of the target layout cellview.

?initCreatePins

Generates all the pins specified in the *I/O Pins* tab of the Update Components and Nets form.

The default is `t`.

`?initGlobalNetPins`

Generates layout pins for the global nets in the schematic, if the *Create* option on the *I/O Pins* tab is selected.

If the *Create* option is not selected, no global pins are created.

The default is `t`.

`?initCreatePadPins`

Generates layout pad pins corresponding to the schematic pins that are connected to the I/O pads, if the *Create* option on the *I/O Pins* tab is selected.

If the *Create* option is not selected, no pad pins are created.

The default is `t`.

`?initCreateInstances`

Generates all the instances in the schematic that do not have one of the ignore properties attached to them.

The default is `t`.

`?initCreateBoundary`

Generates a design boundary using the layer and size information specified in the *Boundary* tab of the Update Components and Nets form.

The default is `t`.

`?initCreateSnapBoundary`

Generates a snap boundary.

The default is `nil`.

`?initDoStacking`

Enables the abutment of ordered lists of MOS transistors into chains when they are generated in the layout.

The default is `nil`.

`?initDoFolding`

Folds transistors into separate fingers to prevent the gate width from exceeding the manufacturing foundry specification size.

The default is `nil`.

`?initCreateMTM`

Preserves user-defined one-to-one, many-to-many, many-to-one, and one-to-many device correspondence defined in the Define Device Correspondence form.

User-defined internal bindings made using the Update Binding form are not preserved, and no report is generated for missing devices or shapes within a mapped group.

The default is `nil`.

`?deleteUnmatchedInsts`

Deletes layout instances that no longer exist in the schematic.

If `?deleteUnmatchedInsts` is set to `nil`, the unmatched instances are not deleted, but they are represented by a marker in the layout.

The default is `nil`.

`?deleteUnmatchedPins`

Deletes layout pins that no longer exist in the schematic.

If `?deleteUnmatchedPins` is set to `nil`, the unmatched pins are not deleted, but they are represented by a marker in the layout.

The default is `nil`.

`?updateReplacesMasters`

Updates existing instances that use an incorrect master to use the correct master.

The default is `t`.

`?updateWithMarkers`

Puts a marker on the instance with the incorrect master and creates a new instance with the correct master, if `?updateReplacesMasters` is set to `t`.

If `?updateReplacesMasters` is set to `nil`, the instance with the incorrect master is removed and replaced at the same location with an instance with the correct master.

The default is `nil`.

`?updateLayoutParameters`

Updates the parameters and parameter values on layout instances to match those on their schematic counterparts.

The default is `nil`.

`?updateNetSigType`

Updates the layout net signal types to match their schematic counterparts.

The default is `t`.

`?UpdateNetMinMaxVoltage`

Updates the minimum and maximum voltages on layout nets to match their schematic counterparts.

The default is `t`.

`?updateNetsOnly`

Updates nets and renames instance mismatches.

Note: This argument overrides all the other optional arguments.

The default is `nil`.

`?virtualHierarchy`

Generates virtual hierarchy for schematic symbols that do not have any corresponding layouts.

The default value is `nil`.

`?skipLeafs`

During an update, skips hierarchical layouts that have no instances.

The default is `nil`.

`?extractSchematic`

Extracts the schematic design, if required, before updating the components and nets.

The default is `t`.

If set to `nil`, the schematic is not extracted and the components and nets are not updated.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
layCV=dbOpenCellViewByType( layLibName layCellName layViewName )
lxHierUpdateComponentsAndNets(layCV)
```

Uses the default options for performing an update of the components and nets. The schematic is determined from the current connectivity reference, which can be set using the SKILL function lxSetConnRef.

```
lxHierUpdateComponentsAndNets(layCV ?netsOnly t)
```

Only updates the nets and instance names mismatches.

lxHierUpdateSchematicParameters

```
lxHierUpdateSchematicParameters(
     d_layoutId
     )
     => t / nil
```

Description

Updates the schematic instance parameters hierarchically for all layout instance masters in the layout.

Arguments

d_layoutId	Database ID of a layout cellview.

Value Retuned

t	The schematic parameters were updated.
nil	The operation failed.

Examples

Updates parameter differences for top-level schematic instances bound to layout instances in the specified design `hierarchy`, `ampString`, or `layout` and hierarchically for all layout masters in the design.

```
lcv = dbOpenCellViewByType("hierarchy" "ampString" "layout" "maskLayout" "r")
lxHierUpdateSchematicParameters(lcv)
```

Related Topics

lxUpdateSchematicParameters

lxHiAbout

```
lxHiAbout(
     )
     => t
```

Description

Opens the product information window for Layout XL, which includes the release number and the copyright information.

Arguments

None

Value Returned

`t`	The window is opened.

lxHiAlign

```
lxHiAlign(
     )
     => t
```

Description

Opens the Align form, in which you can specify how selected objects are to be aligned in the layout window design area.

Arguments

None

Value Returned

`t`	The Align form opened.

lxHiBackAnnotateAllActiveDummies

```
lxHiBackAnnotateAllActiveDummies(
     [ ?schCV d_schCellViewId ]
     [ ?layCV d_layCellViewId ]
     [ ?autoSchExtract { t | nil } ]
     )
     => t
```

Description

Backannotates to schematic the symbols of all the active dummy devices—dummy devices that are not tied to a power or ground net and are not bound in the current layout view. If the schematic and the layout cellview IDs are not supplied, Virtuoso tries to determine them automatically.

You can also choose to backannotate inactive dummy devices—dummy devices that are tied to a power or ground net. To do this, type the following in the CIW:

```
envSetVal("layoutXL" "lxDummyBackAnnotateAll" 'boolean t)
```

For more information on backannotation of dummy devices, see Dummy Instances Backannotation.

Arguments

`?schCV` *`d_schCellViewId`*

Database ID of the schematic cellview to which the dummy is backannotated.

Default: `nil`

`?layCV` *`d_layCellViewId`*

Database ID of the layout cellview from which the dummy is backannotated.

Default: `nil`

`?autoSchExtract`

Automatically extracts the schematic, if required.

Default: `nil`

Value Returned

`t`	The symbols of all the active dummies are backannotated to the schematic view.
`nil`	Backannotation of all active dummies failed.

Example

```
lxHiBackAnnotateAllActiveDummies
(?schCV scv ?layCV lcv ?autoSchExtract t)
```

lxHiBackAnnotateSelectedDummies

```
lxHiBackAnnotateSelectedDummies(
     [ ?schCV d_schCellViewId ]
     [ ?layCV d_layCellViewId ]
     [ ?autoSchExtract { t | nil } ]
     [ ?figs l_figs ]
     )
     => t
```

Description

Backannotates to schematic the symbols of selected active dummy devices—dummy devices that are not tied to a power or ground net and are not bound in the current layout view. If the schematic and the layout cellview IDs are not supplied, Virtuoso tries to determine them automatically.

Note: You can also choose to backannotate inactive dummy devices—dummy devices that are tied to a power or ground net. To do this, type the following in the CIW:

```
envSetVal("layoutXL" "lxDummyBackAnnotateAll" 'boolean t)
```

For more information on backannotation of dummy devices, see Dummy Instances Backannotation.

Arguments

`?schCV` *`d_schCellViewId`*

Database ID of the schematic cellview to which the dummy is backannotated.

Default: `nil`

`?layCV` *`d_layCellViewId`*

Database ID of the layout cellview from which the dummy is backannotated.

Default: `nil`

`?autoSchExtract`

Automatically extracts the schematic, if required.

Default: `nil`

`?figs l_figs`

Figure IDs to be used for backannotation.

If figure IDs are available, these are used for backannotation instead of the selected set.

Default: `nil`

Value Returned

`t`	The symbols of selected active dummies are backannotated to the schematic view.
`nil`	Backannotation of selected active dummies failed.

Example

```
d1 = dbFindAnyInstByName(lcv "|Dummy_I1")
figs = cons(d1 insts)
lxHiBackAnnotateSelectedDummies(?schCV scv ?layCV lcv ?autoSchExtract t ?figs
figs)
```

Batch Mode

```
lxRunCmdInVXL(lcv 'lxHiBackAnnotateSelectedDummies list(?schCV scv ?layCV lcv
?autoSchExtract t ?figs figs))
```

lxHiChain

```
lxHiChain(
     [ w_windowID ]
     )
     => t / nil
```

Description

Opens the Generate Chained Devices form, which you can use to specify the transistors in the design to be chained, and starts the interactive chaining function on the design in the specified window. If you do not specify a window, it operates on the design in the current window.

Arguments

w_windowID	ID of the window containing the components to be chained. Valid Values: Any window ID Default: The current window

Value Returned

t	The Generate Chained Devices form was opened and the command was started.
nil	The Generate Chained Devices form was not opened and the command was not started.

Example

```
lxHiChain(windowID)
```

Opens the Generate Chained Devices form and starts the *Generate Chained Devices* function for the design in the specified *windowID*.

The system prompts you to select the instances to be chained.

lxHiCheck

```
lxHiCheck(
     )
```

Description

Checks XL compliance of the layout when called using the *Connectivity – Check – XL Compliance* command.

Arguments

None

Value Returned

None

Example

```
lxHiCheck()
```

lxHiCheckAllTerminalInterfaces

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

Description

Checks the interface and physical-only status of all the scalar terminals in the currently edited layout cellview. Markers are created in the layout for any mismatches found.

Arguments

None

Value Returned

`t`	The check was successful.
`nil`	The cellview is in read-only mode or the required license is not available.

Example

```
lxHiCheckAllTerminalInterfaces()
```

lxHiClone

```
lxHiClone(
     [ w_windowID ]
     )
     => t / nil
```

Description

Opens the Generate Clones form, which you can use to specify cloning parameters, and starts the interactive cloning function, which lets you replicate a section of the layout in such a way that the new piece of layout material can be placed at more than one location, with each part preserving the hierarchical structure and connectivity of the source. You can clone devices, pins, and (if selected from the layout window) interconnect structures such as wires and paths made of shapes. The interactive cloning function operates on the design in the specified window. If you do not specify a window, the function operates on the design in the current window.

Arguments

w_windowID	ID of the window containing the components to be cloned. Valid Values: Any window ID Default: The current window

Value Returned

t	The Generate Clones form was opened and the command was started.
nil	The Generate Clones form was not opened and the command was not started.

Example

```
lxHiClone(windowID)
```

Opens the Generate Clones form and starts the *Generate Clones* function for the design in the specified *windowID*.

The system prompts you to select the connectivity source for the clone. You can select the connectivity source from either the layout cellview or the schematic or another Layout XL source cellview using single or area selection.

If the structure to be cloned has been routed, you must select the structure from the layout view so that all of the required components and interconnects are selected. Selecting the source from the schematic clones only the components and their relative placement, and not the routing or interconnects.

lxHiConnectInstPin

```
lxHiConnectInstPin(
     )
     => nil
```

Description

Opens the Assign Nets form and starts the *Assign Nets* command, which let you assign instance pins to nets in the design.

Arguments

None

Value Returned

`nil`	The command was canceled.

lxHiConvertMosaicToGroupArray

```
lxHiConvertMosaicToGroupArray(
     [ w_windowId ]
     )
     => t / nil
```

Description

Converts the selected mosaic to a group array. If a mosaic is not selected in the window, it starts the command and prompts you to select the mosaic that you want to convert to a group array.

Arguments

w_windowId	ID of the window in which the command is to be started. If not specified, the command is started in the current window.

Value Returned

`t`	The command is started successfully.
`nil`	The command could not be started.

Example

Converts the selected mosaic to a group array in the current window.

```
lxHiConvertMosaicToGroupArray()
==> Converted 1 mosaic to groupArray.
    t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxHiCreateGroup

```
lxHiCreateGroup(
     [ w_windowID ]
     )
     => t / nil / d_group
```

Description

Creates a figGroup using the selected set of components, if the selected set is not already inside an existing figGroup. If the selected set is inside a virtual hierarchy in a cellview opened in Virtuoso Layout Suite EXL, the SKILL function creates a virtual group.

Arguments

w_windowID	ID of the window containing the selected set for figGroup creation. Valid Values: Any window ID Default: The current window

Value Returned

t	The figGroup is created.
nil	The figGroup could not be created.
d_group	Database ID of the virtual group created if the selected set is inside a virtual hierarchy and the cellview is opened in Virtuoso Layout Suite EXL. If the selected set is not inside a virtual hierarchy in a cellview opened in Virtuoso Layout Suite EXL, a regular figGroup is created.

Example

```
lxHiCreateGroup(currWindowId)
```

Creates a figGroup using the selected components in the specified layout window.

lxHiCreateInstFromSch

```
lxHiCreateInstFromSch(
     [ w_windowID ]
     )
     => t / nil
```

Description

Opens the Generate Selected Components form and starts the interactive command which lets you generate selected schematic components in the layout view. The function operates on the design in the specified window. If you do not specify a window, the function operates on the current window.

Arguments

w_windowID	ID of the window containing the components to be generated. Valid Values: Any window ID Default: The current window

Value Returned

t	The Generated Selected Components form was opened and the *Generate Selected Components* command was started.
nil	The Generated Selected Components form was not opened and the *Generate Selected Components* command was not started.

Example

```
lxHiCreateInstFromSch(windowID)
```

Opens the Generated Selected Components form and starts the interactive command for the design in the specified *windowID*.

The system prompts you to select in the schematic the instances to be generated.

lxHiCreateMPP

```
lxHiCreateMPP(
     [ w_windowID ]
     )
     => nil
```

Description

Opens the Create Multipart Path form, which you can use to specify the parameters for creating a multipart path, and starts the interactive command which lets you create a path in the layout window. The function operates on the design in the specified window. If you do not specify a window, the function operates on the design in the current window.

Arguments

w_windowID	ID of the window containing the design you want to modify. Valid Values: Any window ID Default: The current window

Value Returned

nil	The command was canceled.

Example

```
lxHiCreateMPP(windowID)
```

Opens the Create Multipart Path form and starts the *Create Multipart Path* command for the design in the specified *windowID*.

The system prompts you to click in the layout window to create the new multipart path.

lxHiCreatePinsFromLabels

```
lxHiCreatePinsFromLabels(
     )
     => t
```

Description

Creates pins in the layout from labels for those schematic terminals that do not have a corresponding layout pin. Additionally, creates pins for terminals that already have pins but labels are swapped in the layout.

Arguments

None

Value Returned

`t`	Layout pins are created.

Example

```
lxHiCreatePinsFromLabels()
```

lxHiDefineDeviceCorr

```
lxHiDefineDeviceCorr(
     )
     => t
```

Description

Opens the Define Device Correspondence form, which lets you bind schematic devices to layout devices.

Arguments

None

Value Returned

`t`	The command was run and the changes accepted.

lxHiEditComponentTypes

```
lxHiEditComponentTypes(
     [ w_windowID ]
     )
     => t
```

Description

Opens the Configure Physical Hierarchy window in *Component Types* mode, which lets you define component types and assign cells to them. The function operates on the design in the specified layout window. If you do not specify a layout window, it operates on the design in the current one; if there is no current layout window, the function exits.

Arguments

w_windowID	ID of the layout window containing the design for which you want to define component types. Valid Values: Any layout window ID Default: The current layout window

Value Returned

t	The command was successful.
nil	The command failed because the specified window is not a layout window or the current window is not a layout window.

Example

Opens the Configure Physical Hierarchy window for the design in the specified *windowID*.

```
lxHiEditComponentTypes(windowID)
```

lxHiGenerateSelectedFromLayout

```
lxHiGenerateSelectedFromLayout(
     [ w_windowID ]
     )
     => t | nil
```

Description

Starts the *Generate Selected From Layout* enter function, which lets you generate selected layout components in the corresponding schematic cellview. The layout window from which the components are selected is specified. If the layout window is not specified, the function operates on the current window.

Arguments

w_windowID	ID of the layout window from which the components are selected for generation.

Value Returned

`t`	The command was successful, *Generate Selected From Layout* was started.
`nil`	The command failed because the specified window or the current window is not a layout window.

Example

Starts the interactive *Generate Selected From Layout* command in the specified window. The command prompts the user to select unbound layout instances to be generated in the schematic.

```
lxHiGenerateSelectedFromLayout(windowId)
```

lxHilChain

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

Description

Starts the enter function available through the shortcut menu of selected instances, pins, or a group of abutted instances to support incremental interactive chaining. The selected instances, pins, or a single group of abutted objects are considered as anchors to start the chain and the subsequent user-specified points are then used to interactively chain the instances and top-level pins until the Esc key is pressed to stop the function.

Arguments

None

Value Returned

`t`	The user-defined points to build the interactive chain were specified using mouse clicks.
`nil`	The interactive chain could not be built.

Example

```
lxHiIChain()
```

lxHiLockSelected

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

Description

Sets the placement status of the currently selected objects to `locked`. Each object is locked at its current location and orientation in the layout view.

Tip

Cadence recommends that you change the placement status using the Constraint Manager and Property Editor assistants instead.

Arguments

None

Value Returned

`t`	The placement status of the selected objects was set to `locked`.
`nil`	The placement status could not be changed, typically because there are no objects selected.

lxHiMoveAutomatically

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

Description

Runs the *Place As In Schematic* command, which moves all of the components into the design boundary, maintaining the relative positions of the components in the schematic. If there is no connectivity reference, the Update Connectivity Reference form is opened.

Arguments

None

Value Returned

`t`	The command executed successfully.
`nil`	The command was canceled before it was run.

lxHiProbe

```
lxHiProbe(
     )
     => t
```

Description

Opens the Probe Options form and starts the *XL Probe* command, which lets you select a component in either the layout or schematic and highlights the corresponding component the other cellview of the pair.

Arguments

None

Value Returned

`t`	The Probe Options form was canceled.

lxHiReInitDesign

```
lxHiReInitDesign(
     )
     => t
```

Description

Opens the Generate Layout form, which you can use to specify which layout components are to be generated from the schematic cellview, or clears the existing layout cellview so that you can restart.

Arguments

None

Value Returned

`t`	The Generate Layout form was displayed.

lxHiSetCorrespondence

This function is no longer supported. Use lxHiDefineDeviceCorr instead.

lxHiSetOptions

```
lxHiSetOptions(
     [ x_tabNumber { 1 | 2 | 3 | 4 | 5 | 6 } ]
     )
     => t / nil
```

Description

Opens the Connectivity form. The form opens with the *General* tab on top, unless you specify a different tab using the optional argument.

Arguments

x_tabNumber { 1 | 2 | 3 | 4 | 5 | 6 }

Integer specifying the tab on top when the form opens.
Valid Values: 1, 2, 3, 4, 5, or 6 (where 1 is the *General* tab; 2 is the *Display* tab; 3 is the *Connectivity* tab; 4 is the *Generation* tab; 5 is the *Parameters* tab; and 6 is the *Routing* tab

Default: 1 (*General* tab).

Value Returned

`t`	The form was opened.
`nil`	The form was not opened.

Examples

```
lxHiSetOptions()
```

Opens the Connectivity form with the *General* tab on top.

```
lxHiSetOptions(5)
```

Opens the Connectivity form with the *Parameters* tab on top.

lxHiStack

```
lxHiStack(
     [ w_windowID ]
     )
     => t / nil
```

Description

Opens the Generate Folded Devices form and starts the *Generate Folded Devices* command, which lets you fold a single transistor or an abutted group of MOS transistors into two or more fingers. The command operates on the design in the specified window. If you do not specify a window, it operates on the design in the current window.

No transistor-specific information is shown in the form until you select at least one transistor in the layout window. If you preselected transistors before using the command, the name of first transistor selected is shown in the *Transistor Name* field.

Arguments

w_windowID	ID of the window containing the components to be folded. Valid Values: Any window ID Default: The current window

Value Returned

`t`	The selected devices were folded successfully.
`nil`	The command failed or the form was canceled.

Example

```
lxHiStack(windowID)
```

Opens the Generate Folded Devices form and starts the *Generate Folded Devices* command for the design in the specified *windowID*.

If you did not preselect the instances to be folded, you are prompted to do so now. If you did preselect instances, the requested folding is performed.

lxHiSwap

```
lxHiSwap(
     [ w_windowID ]
     )
```

Description

Swaps instances and figGroups by launching the Swapping Components command. You can pre-select, partial-select, or post-select instances and figGroups. You can use the function to swap by origin with an option to swap with transform, or you can select one of the nine points on a bounding box: lower-left, center-left, upper-left, lower-center, center-center, upper-center, lower-right, center-right, or upper-right.

Arguments

w_windowID	ID of the window, if provided, on which the instances (or figGroups) need to be swapped.

Value Returned

None

Example

```
lxHiSwap()
```

Swaps the selected instances or figGroups in the current window ID.

```
lxHiSwap(hiGetCurrentWindow())
```

Swaps the selected instances or figGroups in the specified window.

Related Topics

Swapping Components

lxHiSwapComps

This function is obsolete, use lxHiSwap instead.

lxHiUnlockSelected

```
lxHiUnlockSelected(
     g_all
     )
     => t / nil
```

Description

Changes the placement status of the selected objects from `locked`, `placed`, or `firm` to `unplaced`.If the original placement status is `unknown` or `unplaced`, this command does not change it.

Note: Objects with a placement status of `unplaced` are not visible in the layout window, so using this command causes the objects in question to disappear from the canvas.

Tip

Cadence recommends that you change the placement status using the Constraint Manager and Property Editor assistants instead.

Arguments

g_all	This is an obsolete argument retained to maintain backward compatibility. Valid Values: `t`, `nil` Default: `nil`

Value Returned

`t`	The placement status of the selected objects was set to `unplaced`.
`nil`	There are no objects selected.

lxHiUpdateAllTerminalInterfaces

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

Description

Updates the scalar terminals in the currently edited layout cellview to correct mismatches in the interface or physical-only status.

Arguments

None

Value Returned

`t`	Mismatches in the interface and physical-only status were removed.
`nil`	The cellview is in read-only mode or the required license is not available.

Example

```
lxHiUpdateAllTerminalInterfaces()
```

lxHiUpdateBinding

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

Description

Displays the Update Binding form.

Arguments

None

Value Returned

`t`	The form was opened.
`nil`	The form was not opened.

Example

```
lxHiUpdateBinding()
```

lxHiUpdateBusTerminals

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

Description

Updates the implicit bus terminals in the layout cellview based on the corresponding explicit bus terminals in the schematic cellview. The update is important to support interoperability of the layout cellviews with the designs created using the Innovus Implementation System.

Arguments

None

Value Returned

`t`	Implicit bus terminals in the layout cellview are updated.
`nil`	The update failed.

Example

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
       lxHiUpdateBusTerminals(cellView)
  )
)
```

Updates the implicit bus terminals of the layout cellview currently open in the layout editing window.

lxHiUpdateCellViewPair

```
lxHiUpdateCellViewPair(
     [ d_cellviewID ]
     )
     => ( { t_source t_lib t_cell t_view [ t_topcell ] | nil } )
```

Description

Starts the *Update Connectivity Reference* command and opens the Update Connectivity Reference form, where you can specify which schematic view provides the connectivity source for the specified layout view. If you do not specify a layout cellview, the function uses the current layout cellview.

Arguments

d_cellviewID	Database ID of the layout cellview for which you want to set the connectivity reference. Default: The current cellview.

Value Returned

t_source	Specifies that the connectivity reference is a `CELLVIEW`.
t_lib	The name of the library that contains the connectivity source cellview.
t_cell	The cell name of the connectivity source cellview.
t_view	The view name of the connectivity source cellview.
t_topcell	The name of the schematic top cell.
`nil`	There is no connectivity reference defined.

Example

```
lxHiUpdateCellViewPair()
=>   ("CELLVIEW" "ether" "adc_cascode_opamp" "schematic")
```

Confirms that the connectivity reference for the current layout view is a schematic cellview called `adc_cascode_opamp` in a library called `ether`.

lxHiUpdateComponentsAndNets

```
lxHiUpdateComponentsAndNets(
     )
     => t
```

Description

Starts the *Update Components and Nets* command, which lets you update the devices and pins in the layout view based on the devices in the schematic cellview.

Arguments

None

Value Returned

`t`	The command was started.

lxHiUpdateLayoutConstraints

```
lxHiUpdateLayoutConstraints(
     )
     => t
```

Description

Runs the *Update Layout Constraints* command, which transfers constraints from the schematic view to the layout view. Constraints that have been created in the schematic but not yet saved are also transferred. Constraints are transferred only to the top-level layout view and are made read-only.

Arguments

None

Value Returned

`t`	The command started successfully.

lxHiUpdateLayoutParameters

```
lxHiUpdateLayoutParameters(
     )
     => t
```

Description

Starts the *Update Layout Parameters* command, which updates the parameters and values in the layout view to match those set in the schematic view.

Arguments

None

Value Returned

`t`	The command started successfully.

lxHiUpdateSchematicParameters

```
lxHiUpdateSchematicParameters(
     )
     => t
```

Description

Starts the *Update Schematic Parameters* command, which updates the parameters and values in the schematic view to match those set in the layout view.

Arguments

None

Value Returned

`t`	The command started successfully.

lxHiUpdateTerminalNetExpressions

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

Description

Corrects mismatches in the terminal net expressions between layout and schematic reported by the *Check Against Source* command.

Arguments

None

Value Returned

`t`	Mismatches reported in the CAS tab of the Annotation Browser assistant were fixed.
`nil`	The cellview is in read-only mode or the required license is not available.

Example

```
lxHiUpdateTerminalNetExpressions()
```

lxHiVerifyDesign

```
lxHiVerifyDesign(
     )
     => t
```

Description

Starts the *Check Against Source* command and opens the Check Against Source dialog where you can specify how Layout XL is to check the current layout implementation against corresponding schematic view. When you accept the dialog, the command checks the schematic cellview for any changes made since the layout cellview was last changed. It also opens an Info window detailing the differences between the views.

Arguments

None

Value Returned

`t`	The *Check Against Source* command was executed successfully.

lxIsGroupArray

```
lxIsGroupArray(
     d_figId
     )
     => t / nil
```

Description

Checks whether the given object is a group array.

Arguments

d_figId	Database ID of the object that is to be checked.

Value Returned

`t`	The given object is a group array.
`nil`	The given object is not a group array.

Example

The following code checks whether the object whose database ID is stored in `groupArray1` is a group array.

```
lxIsGroupArray(groupArray1)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxIsGroupArrayEnabled

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

Description

Checks whether the group array functionality is available.

Arguments

None

Value Returned

`t`	The group array functionality is available.
`nil`	The group array functionality is not available.

Example

The following code checks whether the group array functionality is available.

```
lxIsGroupArray()
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxIsGroupArrayLocked

```
lxIsGroupArrayLocked(
     d_figId
     )
     => t / nil
```

Description

Checks whether the given group array is locked.

Arguments

d_figId	Database ID of the group array.

Value Returned

`t`	The given group array has the locked attribute set.
`nil`	The given group array is not a locked group array.

Example

The following code checks whether the group array whose database ID is stored in `groupArray1` is locked.

```
lxIsGroupArrayLocked(groupArray1)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxIsGroupArrayRegular

```
lxIsGroupArrayRegular(
     d_figId
     )
     => t / nil
```

Description

Checks whether the given group array is a regular group array. For a regular group array, the number of cells can be modified by updating the number of rows or columns. For a non-regular group array, the number of cells is fixed.

Arguments

d_figId	Database ID of the group array.

Value Returned

`t`	The given group array is a regular group array.
`nil`	The given group array is not a regular group array.

Example

The following code checks whether the group array whose database ID is stored in `groupArray1` is a regular group array.

```
lxIsGroupArrayRegular(groupArray1)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxHiPRBoundaryInteriorHalo

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

Description

Displays an options form to specify if and on which layer a PR boundary interior halo shape should be created during when the *Generate All From Source* command is run.

Arguments

None

Value Returned

`t`	Displays the options form for specifying information related to PR boundary interior halo shape.
`nil`	The operation failed.

Examples

```
lxHiPRBoundaryInteriorHalo()
> t
```

lxLaunchLayoutEXL

```
lxLaunchLayoutEXL(
     [ w_windowID ]
     )
     => t
```

Description

Launches Layout EXL from a specified schematic window. If you do not specify a window ID, Layout EXL is launched from the window returned by the `hiGetCurrentWindow()` function, but only if it is a schematic window.

Arguments

w_windowID	ID of the schematic window from which Layout EXL is to be launched.

Value Returned

`t`	Layout EXL was launched.
`nil`	Layout EXL did not launch.

Example

```
lxLaunchLayoutEXL()
```

lxLaunchLayoutXL

```
lxLaunchLayoutXL(
     [ w_windowID ]
     )
     => t
```

Description

Launches Layout XL from a specified schematic window. If you do not specify a window ID, Layout XL is launched from the window returned by the `hiGetCurrentWindow()` function, but only if it is a schematic window.

Arguments

w_windowID	ID of the schematic window from which Layout XL is to be launched.

Value Returned

`t`	Layout XL was launched.
`nil`	Layout XL did not launch.

```
lxLaunchLayoutXL()
```

lxLaunchLayoutMXL

```
lxLaunchLayoutMXL(
     [ w_windowID ]
     )
     => t
```

Description

Launches Layout MXL from a specified schematic window. If you do not specify a window ID, Layout MXL is launched from the window returned by the `hiGetCurrentWindow()` function, but only if it is a schematic window.

Arguments

w_windowID	ID of the schematic window from which Layout MXL is to be launched.

Value Returned

`t`	Layout MXL was launched.
`nil`	Layout MXL did not launch.

Examples

```
lxLaunchLayoutMXL()
```

lxMakeDummy

```
lxMakeDummy(
     t_netName
     )
     => t / nil
```

Description

Starts the Make Dummy enter function for the selected set of instances which can include mosaics. The function requires a reference point and a position for the created dummy devices. The dummy devices are tied to the named net passed in via `netName`.

Argument

t_netName	Name of the net from which the dummy is to be created.

Value Returned

`t`	A dummy device is created.
`nil`	A dummy device is not created. A warning message is displayed.

Example

```
lxMakeDummy("VSS")
```

Starts the Make Dummy enter function for the net `VSS`. The dummy device can then be created at the target location indicated on the canvas.

Related Topics

Creating Dummy Instances

lxMakePrBoundarySelectable

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

Description

Makes the place and route boundary in the current layout selectable.

Arguments

None

Value Returned

`t`	The place and route boundary is selectable.
`nil`	The command failed.

lxMakePrBoundaryUnselectable

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

Description

Makes the place and route boundary in the current layout unselectable.

Arguments

None

Value Returned

`t`	The place and route boundary is not selectable.
`nil`	The command failed.

lxPermPermutePins

```
lxPermPermutePins(
     )
     => t
```

Description

Starts the *Permute Pins* command, which lets you swap the connectivity or net connections of the pins of a component. You specify the pins you want to permute by selecting them when prompted by the function. The pins to be permuted must belong to different nets and must first be defined as permutable terminals using the `permuteRule` property for the device.

Arguments

None

Value Returned

`t`	The command started successfully.

lxProbeRemoveAll

```
lxProbeRemoveAll(
     )
     => t
```

Description

Removes all probes from the layout cellview.

Arguments

None

Value Returned

`t`	All probes were removed successfully.

lxRegMasterDiffName

```
lxRegMasterDiffName(
     S_function
     )
     => t / nil
```

Description

Registers a SKILL function that can return a name for the master difference reported by the *Check Against Source* command.

Argument

S_function Name of the SKILL function to be registered.

The SKILL function must take the following two arguments and return a string.

- `master`: Database cellview ID of the current layout instance master.
- `expected`: String value representing the expected master in the format `lib/cell/{view1 view2}`.

  The master may have one or multiple views defined in CPH.

Value Returned

`t`	The SKILL function was registered.
`nil`	The SKILL function registration failed.

Example

```
lxRegMasterDiffName('testMasterDiffName)
```

Where the SKILL function `testMasterDiffName` is defined as:

```
procedure(testMasterDiffName1(master expected)
    let((_mbbox _cv _tok _lib _cell _view _views (_name "")
         cH nH cW nW _nbbox)

        _tok = parseString(expected "/")
        _lib = car(_tok)
```

```
_cell = cadr(_tok)
_views = caddr(_tok)
_view = substring(_views 2 (strlen(_views) - 2))
_tok = parseString(_view " ")
_view = car(_tok)

_cv = dbOpenCellView(_lib _cell _view)
when(_cv
    _mbbox = master~>bBox

    cH = testBoxDimY(_mbbox)
    cW = testBoxDimX(_mbbox)

    _nbbox = _cv~>bBox

    nH = testBoxDimY(_nbbox)
    nW = testBoxDimX(_nbbox)

    cond(
        (nH > cH && nW > cW
           _name = "Taller and wider"
        )
        (nH < cH && nW < cW
           _name = "Shorter and thiner"
        )
        (testEqualBasic(nH cH) && nW > cW
           _name = "Wider"
        )
        (testEqualBasic(nH cH) && nW < cW
           _name = "Thinner"
        )
        (nH > cH && testEqualBasic(cW nW)
           _name = "Taller"
        )
        (nH < cH && testEqualBasic(cW nW)
           _name = "Shorter"
        )

    )
    dbClose(_cv)
)
```

```
        _name
    )
)
```

lxRegPostUpdateComponentsAndNets

```
lxRegPostUpdateComponentsAndNets(
     S_function
     )
     => t / nil
```

Description

Registers a SKILL function to be used as a post-function hook for `lxUpdateComponentsAndNets` to enable replacement of pins that have their layer and width characteristics defined using the Design Intent tool. `lxRegPostUpdateComponentsAndNetsFn`, when used in combination with `lxRegPostUpdateComponentsAndNetsFn`, avoids the need for deleting the pins defined in Design Intent before the pins can be updated.

Argument

S_function	Name of the SKILL function to be registered.

Value Returned

`t`	The SKILL function was registered.
`nil`	The SKILL function registration failed.

Example

```
> procedure(testUpdateComponentsAndNetsPostHook(scv lcv) printf("Test UCN SKILL
PostHook %s %s\n" scv~>viewName lcv~>viewName))
=> testUpdateComponentsAndNetsPostHook
> lxRegPostUpdateComponentsAndNetsFn("testUpdateComponentsAndNetsPostHook")
=> t
> lxHiUpdateComponentsAndNets()
```

Runs a test to validate the post-function SKILL hook set for the `lxUpdateComponentsAndNets` SKILL function.

lxRegPrePosition

```
lxRegPrePosition(
     t_lib
     t_cell
     t_view
     s_skillFunction
     )
     => t / nil
```

Description

Registers a SKILL function for a schematic Pcell to be called for newly generated layout instances corresponding to each instance of the schematic Pcell when *Generate All From Source*, *Generate Selected from Source*, or U*pdate Components and Nets commands* are run. Registration is done once before generation. The registered SKILL function is called after instance generation but before positioning.

Only one SKILL function can be registered per schematic Pcell.

Arguments

t_lib	Name of a library.
t_cell	Name of a cell.
t_view	Name of a view.
s_skillFunction	Name of the SKILL function that takes the argument of the instances generated.

Value Returned

t	The function is registered successfully.
nil	The function is not registered.

Examples

```
procedure(invPrePosition(instances))
```

Chain all the instances.

```
lxChain(instances list(leftEdge(car(instances)) bottomEdge(car(instances))
?useDeviceOrder t)
```

```
   => t
)
```

Registers the SKILL function `InvPrePosition` for a schematic Pcell with library `schLib`, cell name `inv`, and view name `schematic`.

```
lxRegPrePosition("schLib" "inv" "schematic" 'InvPrePosition)
=> t
```

The following example, calls a registered function with the cellview.

```
lxRegPrePosition("" "" "" 'preFunc)
```

Related Topic

lxChain

lxUnregPrePosition

lxRegPreUpdateComponentsAndNets

```
lxRegPreUpdateComponentsAndNets(
     S_function
     )
     => t / nil
```

Description

Registers a SKILL function to be used as a pre-function hook for `lxUpdateComponentsAndNets` to enable replacement of pins that have their layer and width characteristics defined using the Design Intent tool. `lxRegPreUpdateComponentsAndNetsFn`, when used in combination with `lxRegPostUpdateComponentsAndNetsFn`, avoids the need for deleting the pins defined in Design Intent before the pins can be updated.

Argument

S_function	Name of the SKILL function to be registered.

Value Returned

`t`	The SKILL function was registered.
`nil`	The SKILL function registration failed.

Example

```
> procedure(testUpdateComponentsAndNetsPreHook(scv lcv) printf("Test UCN SKILL
PreHook %s %s\n" scv~>viewName lcv~>viewName))
=> testUpdateComponentsAndNetsPreHook
> lxRegPreUpdateComponentsAndNetsFn("testUpdateComponentsAndNetsPreHook")
=> t
> lxHiUpdateComponentsAndNets()
```

Runs a test to validate the pre-function SKILL hook set for the `lxUpdateComponentsAndNets` SKILL function.

lxRemoveSynchronousCloneFromFamily

```
lxRemoveSynchronousCloneFromFamily(
     [ cloneList ]
     )
     => t / nil
```

Description

Removes a specified synchronous clone from its family. As a result, the clone is desynchronized and it behaves as a basic group. If a clone that belongs to a two-member synchronous family is removed from the family, the other member clone is also removed and the family is deleted.

Argument

cloneList The cloneList can be:

- A single synchronous clone
- A list of synchronous clones
- Nothing specified

Note:

- When the function is run with no argument specified, the function is called on selected figures, such as, when calling `lxRemoveSynchronousCloneFromFamily(geGetSelSet())`
- If the argument list contains a non-synchronous clone object, the object is ignored.

Value Returned

`t`	At least one clone has been removed from the clone family.
`nil`	No clone has been desynchronized.

Example

```
lxRemoveSynchronousCloneFromFamily (cloneList)
```

lxRunCmdInVXL

```
lxRunCmdInVXL(
     d_layCV
     u_cmd
     g_cmdArgs
     [ ?schCV d_schCV ]
     )
     => t / nil
```

Description

This function is being maintained only to support backward compatibility. Cadence recommends that you use lxRunCmdInXL instead. If you were to use `lxRunCmdInVXL`, use it to run a user-supplied command in Layout XL, with CPH and the Binder enabled, without opening the windows. The PhysConfig used is obtained from the layout cellview. If you want to work in Automatic mode, the schematic cellview, an optional argument, is required. The batch function does not work with the following: Interactive editing, Incremental binding, and Incremental extraction.

Arguments

d_layCV	ID of the layout cellview.
u_cmd	The command symbol.
g_cmdArgs	The list of command arguments. If no arguments are required, provide *nil*.
?schCV *d_schCV*	A schematic cellview ID to be used to run the command in Automatic mode. This is an optional argument.

Value Returned

t	Command ran successfully.
nil	Error in command.
Return value from supplied command	Return value based on the SKILL command provided by the user.

Example

```
lxRunCmdInVXL(lcv 'ciLxComparisonReport list(?layoutCV lcv ?showReport nil
?useViewNames t ?useTimeStamp t))
```

Result

```
lxRunCmdInVXL(lcv 'bndGetBoundObjects list(inst1) ?schCV scv)
```

lxRunCmdInXL

```
lxRunCmdInXL(
     d_layCV
     u_cmd
     [ ?cmdArgs g_cmdArgs ]
     [ ?hierarchical {t | nil} ]
     [ ?editLayout {t | nil} ]
     [ ?editSchematic {t | nil} ]
     [ ?skipLeafs {t | nil} ]
     [ ?noSchematic {t | nil} ]
     )
     => t / nil
```

Description

Runs a user-defined command in Layout XL in the current layout or for each unique layout master in the specified layout cellviews hierarchy. When the `lxRunCmdInXL` SKILL function is run, the CPH and the binder are enabled, but no windows open. The physConfig and the schematic connectivity reference is obtained from the layout cellviews by setting the SKILL functions lxSetConnRef and lxSetConfigRef. The `lxRunCmdInXL` function does not support the following commands: interactive edits, incremental binding, and incremental extraction. The user-supplied command must have at least two arguments—the schematic cellview ID and the layout cellview ID for the Layout XL session.

Arguments

d_layCV	ID of the layout cellview.
u_cmd	The command symbol.
?cmdArgs *g_cmdArgs*	The list of additional command arguments.
?hierarchical	The user-defined command is first run alphabetically on each unique layout master in the specified layout cellview. Then, run on the specified layout cellview, provided the cellview can be opened in Layout XL. The default is `nil`, which runs the user command only on the specified layout cellview.
?editLayout	Opens the layout cellviews in append mode to enable edits to the layouts. The layout cellviews are left open after the command is run to enable you to save the edited layouts, if appropriate. The default is `nil`, which opens the layout cellviews in read-only mode.
?editSchematic	The default `nil` opens the schematic cellviews in read-only mode. When set to `t`, the schematic cellviews are opened in append mode to enable edits to the schematic. Any edited schematic cellviews are left open after the command is run to enable you to save the schematics, if required. The default is `nil`.
?skipLeafs	Skips any layouts with no instances and does not run an XL session on the layout. The default is `nil`.
?noSchematic	Specifies whether `lxRunCmdInXL` should run without the schematic. The default is `nil`.

Value Returned

Return value from supplied command	Return value based on the SKILL command provided by the user.

`nil`	Error in command.

Example

```
lxRunCmdInXL(lcv 'lxCheckAgainstSource)
```

Result

```
= lxRunCmdInXL(lcv 'lxUpdateComponentsAndNets ?hierarchical t ?editLayout t)
procedure(myFn(scv lcv doit)
   when(doit
       when(myUpdateSchematicFn(scv lcv)
           dbSave(scv)
       )
   )
)

lxRunCmdInXL(lcv 'myFn ?hierarchical t ?cmdArgs list(doit) ?editSchematic t)
```

lxSelectedDeleteNetRouting

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

Description

Deletes routing of the selected nets.

Arguments

None

Value Returned

`t`	Routing on the selected nets is deleted.
`nil`	The routing on nets is not deleted.

Example

```
lxSelectedDeleteNetRouting ()
```

lxSelectedExtendChain

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

Description

Extends the current selection to all the instances abutted to a selected instance such that the entire chain is selected.

Arguments

None

Value Returned

`t`	The selection has been extended to an abutted instance that was not already selected.
`nil`	No new abutted instance was found. The selection could not be extended.

Example

If instance `I1` from the chain; `I1`, `I2`, `I3`, `I4`; is selected, calling `lxSelectedExtendChain()` will add `I2`, `I3` and `I4` to the selected set.

lxSelectedExtendExpanded

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

Description

Extends the current selection to any related instances, terminals, and nets.

The function expands selection to any:

- Unselected instances that are part of a folded, mfactored, sfactored, vectored, or lxCombination device in the current selected set.
- Bus or bundle nets or terminals.
- Instances, terminals, or nets that share the same base name if the binder is not present.

Arguments

None

Value Returned

`t`	The selection is extended successfully.
`nil`	No new instance, terminal, or net was found or the selection could not be extended.

Example

Consider that an Mfactor device has two instances `M1` and `M2` and a bus pin has pins `P1` and `P2`. If `M1` and `P1` are selected and `lxSelectedExtendExpanded` is run, then `M2` and `P2` are also added to the selected set.

lxSelectedExtendSchematic

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

Description

Extends the current selection of instances to include any instances that are part of the bound top-level schematic.

Arguments

None

Value Returned

`t`	The selection was extended.
`nil`	No new instance was found or the selection could not be extended.

Example

Consider that `NAND2_0|N0` is selected while `NAND2_0|N1`, `NAND2_0|P0`, and `NAND2_0|P3` are not selected but are bound to the same top-level schematic instance as `NAND2_0|N0`.

When you run `lxSelectedExtendSchematic()`, `NAND2_0|N1`, `NAND2_0|P0`, and `NAND2_0|P3` are also added to the selected set.

lxSelectedExtendSelection

This function is now deprecated. Use `lxSelectedExtendExpanded`.

lxSelectedLock

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

Description

Locks the nets currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The nets were locked.
`nil`	The nets were not locked.

Example

```
lxSelectedLock()
```

lxSelectedLockNet

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

Description

Locks the nets attached to the shapes currently selected in the layout cellview

Arguments

None

Value Returned

`t`	The nets were locked.
`nil`	The nets were not locked.

Example

```
lxSelectedLockNet()
```

lxSelectedRemoveIgnore

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

Description

Removes the ignore property on selected instances and pins.

Arguments

None

Value Returned

`t`	The ignore property was removed.
`nil`	The ignore property was not removed.

Example

```
lxSelectedRemoveIgnore()
```

lxSelectedRemoveIgnoreForCheck

This function is no longer supported. Use lxSelectedRemoveIgnore instead.

lxSelectedRoute

```
lxSelectedRoute(
     g_useWAOverrides
     )
     => t / nil
```

Description

Routes the nets currently selected in the layout cellview.

Arguments

g_useWAOverrides	Controls whether Wire Assistant constraint overrides are used for routing the selected nets.

Value Returned

`t`	Selected nets were routed successfully.
`nil`	One or more selected nets were not completely routed.

Example

```
lxSelectedRoute(t)
```

Routes the selected nets using the Wire Assistant constraint overrides.

```
lxSelectedRoute(nil)
```

Routes the selected nets using the default constraint lookup.

lxSelectedRouteNet

```
lxSelectedRouteNet(
     g_useWAOverrides
     )
     => t / nil
```

Description

Routes the nets attached to the shapes currently selected in the layout cellview.

Arguments

g_useWAOverrides	Controls whether Wire Assistant constraint overrides are used for routing the nets attached to selected shapes.

Value Returned

`t`	Nets attached to currently selected shapes were successfully routed.
`nil`	One or more nets attached to the currently selected shapes were not routed successfully.

Example

```
lxSelectedRoute(t)
```

Routes the nets attached to selected shapes using the Wire Assistant constraint overrides.

```
lxSelectedRoute(nil)
```

Routes the nets attached to the selected shapes using the default constraint lookup.

lxSelectedSelectAttachedGlobalNets

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

Description

Selects the power, ground, or global nets attached to the objects currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectAttachedGlobalNets()
```

lxSelectedSelectAttachedNets

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

Description

Selects the nets attached to the objects currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectAttachedNets
```

lxSelectedSelectAttachedSignalNets

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

Description

Selects nets other than power, ground, or global nets attached to the objects currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectAttachedSignalNets()
```

lxSelectedSelectExternalGlobalNets

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

Description

Selects any power, ground, or global nets that are externally connected to the groups, clones, or modgens currently selected in the layout cellview. An external net connects an object within a group to an object outside the group.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectExternalGlobalNets()
```

lxSelectedSelectExternalNets

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

Description

Selects the external nets for the groups, clones, or modgens currently selected in the layout cellview. An external net connects an object within a group to an object outside the group.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectExternalNets()
```

lxSelectedSelectExternalSignalNets

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

Description

Selects any nets other than power, ground, or global nets that are externally connected to the groups, clones, or modgens currently selected in the layout cellview. An external net connects an object within a group to an object outside the group.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectExternalSignalNets()
```

lxSelectedSelectInternalGlobalNets

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

Description

Selects any power, ground, or global nets that are internal to the groups, clones, or modgens currently selected in the layout view. An internal net connects objects contained within a group.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectInternalGlobalNets()
```

lxSelectedSelectInternalNets

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

Description

Selects the internal nets for the groups, clones, or modgens currently selected in the layout view. An internal net connects objects contained within a group.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectInternalNets()
```

lxSelectedSelectInternalSignalNets

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

Description

Selects any nets other than power, ground, or global nets that are internal to the groups, clones, or modgens currently selected in the layout view. An internal net connects objects contained within a group.

Arguments

None

Value Returned

`t`	The nets were selected.
`nil`	The nets were not selected.

Example

```
lxSelectedSelectInternalSignalNets()
```

lxSelectedSelectNetsShapes

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

Description

Selects all the shapes attached to the nets currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The shapes were selected.
`nil`	The shapes were not selected.

Example

```
lxSelectedSelectNetsShapes()
```

lxSelectedUnlock

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

Description

Unlocks the nets currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The nets were unlocked.
`nil`	The nets were not unlocked.

Example

```
lxSelectedUnlock()
```

lxSelectedUnlockNet

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

Description

Unlocks the nets attached to the shapes currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The nets were unlocked.
`nil`	The nets were not unlocked.

Example

```
lxSelectedUnlockNet()
```

lxSelectedUpdateFromSource

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

Description

Updates the masters, connectivity, and parameters of the instances currently selected in the layout cellview.

Arguments

None

Value Returned

`t`	The instances were updated.
`nil`	The instances were not updated.

Example

```
lxSelectedUpdateFromSource()
```

lxSelectedUpdateIgnore

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

Description

Sets the ignore property on selected instances and pins.

Arguments

None

Value Returned

`t`	The ignore property was set.
`nil`	The ignore property was not set.

Example

```
lxSelectedUpdateIgnore()
```

lxSelectedUpdateIgnoreForCheck

This function is no longer supported. Use lxSelectedUpdateIgnore instead.

lxSelectSynchronousFamily

```
lxSelectSynchronousFamily(
     [ cloneList ]
     )
     => t / nil
```

Description

Selects all the clones in the layout canvas and the Navigator that are members of the specified clone family. If all the family members are already selected, no more clones are selected on the canvas or in the Navigator. If any new clones are identified, they are selected on the canvas and in the Navigator.

Arguments

cloneList	Represents a single synchronous clone or a list of selected figures, which are synchronous clones. If the selected set of figures contains any non-synchronous clones, they are ignored.

Value Returned

`t`	Atleast one clone is present in the specified synchronous clone family.
`nil`	No synchronous clones are present.

Example

```
lxSelectSynchronousFamily (cloneList)
```

lxSetAreaEstimationOptions

```
lxSetAreaEstimationOptions(
     [ ?source1 t_source1 ]
     [ ?source1Val source1Val ]
     [ ?source2 t_source2 ]
     [ ?source2Val source2Val ]
     [ ?calcMethod t_calcMethod ]
     [ ?areaCalc t_areaCalc ]
     )
     => t / nil
```

Description

Sets the area estimation options to be used in generating a rectangular boundary in either the Generate Layout or Update Components and Nets form. The function is valid only if you specify a rectangular boundary using the `lxSetBoundaryOptions` function. It must be called between calls to `lxGenerateStart` and `lxGenerateFinish` or `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`.

Arguments

`?source1`	Specifies the first parameter that defines how the boundary is to be calculated. Valid Values: `Utilization`, `AspectRatio`, `BoundaryWidth`, `BoundaryHeight` Default: `Utilization`
`?source1Val`	Value for the parameter specified by `?source1`. Default: `20`
`?source2`	Specifies the second parameter that defines how the boundary is to be calculated. The value must be different from the one specified by `?source1`. Valid Values: `Utilization`, `AspectRatio`, `BoundaryWidth`, `BoundaryHeight` Default: `AspectRatio`
`?source2Val`	Value for the parameter specified by `?source2`. Default: `1`
`?calcMethod`	Specifies the calculation method to be used. Valid Values: `Internal`, `UserDefined` Default: `Internal`
`?areaCalc`	Specifies the area calculation function to use. Valid Values: `PR Boundary`, `BBox`, *`t_funcName`* Default: `PR Boundary` When `?calcMethod` is `UserDefined`, specify the name of a user-defined function, *`t_funcName`*, to use.

Value Returned

`t`	Area estimation options were set.
`nil`	Area estimation options were not set.

Examples

```
lxGenerateStart(schCv layCv (?extractSchematic t))
  lxSetAreaEstimationOptions(
    ?source1 "Utilization"
    ?source1Val 20
    ?source2 "AspectRatio"
    ?source2Val 1
    ?calcMethod "Internal"
    ?areaCalc "PR Boundary")
lxGenerateFinish(schCv layCv (?extractSchematic t))
```

Generates a layout.

```
lxUpdateComponentsAndNets(schCv layCv (?extractSchematic t))
  lxSetAreaEstimationOptions(
    ?source1 "Utilization"
    ?source1Val 20
    ?source2 "AspectRatio"
    ?source2Val 1
    ?calcMethod "Internal"
    ?areaCalc "PR Boundary")
lxUpdateComponentsAndNetsFinish(schCv layCv (?extractSchematic t))
```

Updates an existing layout.

lxSetBoundaryOptions

```
lxSetBoundaryOptions(
     [ ?shape t_shape ]
     [ ?origin g_origin ]
     [ ?points lg_points ]
     )
     => t / nil
```

Description

Sets the boundary options in the Generate Layout and Update Components and Nets forms. It must be called between calls to `lxGenerateStart` and `lxGenerateFinish` or `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`.

When used with the other `lxSet*` functions, this provides an alternative interface to the Generate Layout or Update Components and Nets forms. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

`?shape`	Specifies the shape of the boundary. Valid Values: `Rectangle` and `Polygon` Default: `Rectangle`
`?origin`	Specifies the coordinates of the boundary's origin. This argument is required when `?shape` is set to `Rectangle`.
`?points`	List of points defining the vertices of a polygonal boundary. This argument is required when the `?shape` is set to `Polygon`.

Value Returned

`t`	Boundary options were set.
`nil`	Boundary options were not set.

Examples

```
lxGenerateStart(schCv layCv (?extractSchematic t))
  lxSetBoundaryOptions(
    ?shape "Rectangle"
    ?origin 0:0)
lxGenerateFinish(schCv layCv (?extractSchematic t))
```

Creates a rectangular boundary with its origin at (0, 0) during layout generation.

```
lxUpdateComponentsAndNetsStart(schCv layCv (?extractSchematic t))
  lxSetBoundaryOptions(
    ?shape "Rectangle"
    ?origin 0:0)
lxUpdateComponentsAndNetsFinish(schCv layCv (?extractSchematic t))
```

Updates an existing layout.

lxSetCloneFamilyName

```
lxSetCloneFamilyName(
     clone familyName
     )
     => t / nil
```

Description

Sets a new name for the family to which the specified clone belongs.

Arguments

clone	The clone for which the family name is set.
familyName	New name for the clone family.

Value Returned

t	The clone family name was set.
nil	The specified object is not a synchronous clone or the name specified for the clone family is invalid.

Example

```
lxSetCloneFamilyName (clone familyName)
```

lxSetConfigRef

```
lxSetConfigRef(
     t_layLib
     t_layCell
     t_layView
     t_referenceType
     [ ?cfgLib t_cfgLib ]
     [ ?cfgCell t_cfgCell ]
     [ ?cfgView t_cfgView ]
     [ ?closeLay g_closeLay  ]
     )
     => t / nil
```

Description

Sets or unsets the physical configuration cellview for a specified layout cellview. You specify the layout cellview in question and whether the reference type is `NONE` or `physConfig`. If the reference type is `physConfig`, specify the library, cell, and view names of the physical configuration to use.

Arguments

t_layLib	Name of the library in which the layout cellview is stored.
t_layCell	Name of the cell for which you want to set the connectivity reference.
t_layView	Name of the view for which you want to set the connectivity reference.
t_referenceType	Specifies whether or not a configuration reference is set for the layout cellview. Valid Values: `NONE`, `physConfig` If you set this option to `physConfig`, use the *cfgLib*, *cfgCell*, and *cfgView* arguments to specify the physical configuration view to use.
`?cfgLib` *t_cfgLib*	Name of the library in which the physical configuration is stored.
`?cfgCell` *t_cfgCell*	Name of the cell to use as the configuration reference.

`?cfgView` *`t_cfgView`*

Name of the view to use as the configuration reference.

`?closeLay` *`g_closeLay`*

A Boolean specifying whether currently open layout cellview should be closed. The default is `nil`, which means the layout cellview stays open.

Value Returned

`t`	The physical configuration reference was set.
`nil`	The physical configuration reference was not set.

Example

```
lxSetConfigRef("libA" "cellA" "layout" "NONE"
```

Sets the physical configuration cellview for a layout cellview `cellA` to `NONE`, which means that no physical configuration is set.

```
lxSetConfigRef("libA" "cellA" "layout" "physConfig" ?cfgLib "libB" ?cfgCell "cell
A" ?cfgView "physConfig")
```

Sets the physical configuration cellview for layout cellview `libA`, `cellA`, `layout` to be of type `physConfig`, where the configuration is stored in the library `libB`, cell `cellA`, and view `physConfig`.

lxSetConnRef

```
lxSetConnRef(
     t_layLib
     t_layCell
     t_layView
     t_sourceType { CELLVIEW | NONE }
     [ ?schLib t_schLib ]
     [ ?schCell t_schCell ]
     [ ?schView t_schView ]
     [ ?closeLay g_closeLay ]
     )
     => t / nil
```

Description

Sets or unsets the connectivity reference for a specified layout cellview without opening the layout view graphically. You must specify the layout view and whether the source type is `NONE` or `CELLVIEW`. If the source type is `CELLVIEW`, you can specify the schematic cellview to use (by default it is set to the schematic view in the same library and cell). The function also saves the layout cellview, but only if had not been modified prior to `lxSetConnRef` being called.

Arguments

t_layLib

Name of the library in which the layout cellview is stored.

t_layCell

Name of the cell for which you want to set the connectivity reference.

t_layView

Name of the view for which you want to set the connectivity reference.

t_sourceType

The type of connectivity reference for the design.
Valid Values: `NONE`, `CELLVIEW`

When set to `CELLVIEW`, you can use the *schLib*, *schCell*, and *schView* arguments to specify the schematic view to use. By default it is set to the schematic view in the same library and cell.

`?schLib` *`t_schLib`*

Name of the library in which the connectivity reference is stored.

`?schCell` *`t_schCell`*

Name of the cell you want to use as a connectivity reference.

`?schView` *`t_schView`*

Name of the view you want to use as a connectivity reference.

`?closeLay` *`g_closeLay`*

A Boolean specifying whether currently open layout cellview should be closed. The default is `nil`, which means the layout cellview stays open.

Value Returned

`t`	The connectivity reference was set.
`nil`	The connectivity reference was not set.

Example

```
lxSetConnRef( "libA" "cellA" "layout" "NONE" )
```

Unsets the connectivity reference for layout cellview `cellA`.

```
lxSetConnRef( "libA" "cellA" "layout" "CELLVIEW" ?schLib "libB" ?schCell "cellA"
?schView "schematic" )
```

Sets the connectivity reference for layout cellview `cellA` in `libA` to schematic cellview `cellA` in library `libB`.

lxSetGenerateOptions

```
lxSetGenerateOptions(
     [ ?pins { t | nil } ]
     [ ?instances { t | nil } ]
     [ ?boundary { t | nil } ]
     [ ?snapBoundary { t | nil } ]
     [ ?stacks { t | nil } ]
     [ ?folds { t | nil } ]
     [ ?posMinSep n_posMinSep ]
     [ ?inBoundary { t | nil } ]
     [ ?mtm { t | nil } ]
     [ ?extract { t | nil } ]
     [ ?virtualHierarchy { t | nil } ]
     )
     => t
```

Description

Tells *Generate All From Source* or *Update Components And Nets* what types of objects to generate. You can call it between calls to `lxGenerateStart` and `lxGenerateFinish` or `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`.

When used with the other `lxSet*` functions, this provides an alternative interface to the Generate Layout and Update Components and Nets forms. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

`?pins`	Generates all the pins and snaps them to the placement grid.
`?instances`	Generates all the instances in the schematic that do not have one of the ignore properties attached to them.
`?boundary`	Generates a place and route boundary based on the settings you make in lxSetBoundaryOptions and, where applicable, lxSetAreaEstimationOptions.
`?snapBoundary`	Generates a rectangular snap boundary that encloses the generated PR boundary. You can generate a snap boundary only if `?boundary` is set to `t`.
`?stacks`	Automatically abuts MOS transistors into chains during layout generation.
`?folds`	Automatically splits devices into fingers to prevent gate width from exceeding a specified size.
`?posMinSep`	Defines the minimum separation between instances and pins.
`?inBoundary`	Positions the instances and pins inside the prBoundary.
`?mtm`	Preserves user-defined bindings of devices between the schematic and the layout. **Note:** This option preserves user-defined one-to-one, many-to-many, many-to-one, and one-to-many device correspondence defined in the Define Device Correspondence form.
`?extract`	Extracts the connectivity of the design after layout generation is complete.
`?virtualHierarchy`	Generates virtual hierarchy for schematic symbols with no corresponding layouts.

Value Returned

`t`	The layout generation options were set.

`nil`	The layout generation options were not set.

Examples

```
lxGenerateStart(schId layId)
  lxSetGenerateOptions(
    ?pins t
    ?instances t
    ?boundary t
    ?snapBoundary t
    ?stacks t
    ?folds t
    ?mtm t
    ?extract t
    ?virtualHierarchy t)
lxGenerateFinish(schId layId)
```

Generates a layout including pins, instances, place and route and snap boundaries, chains and folds the transistors that are generated and assigns many-to-many bindings. It then runs extraction after layout generation and generates a virtual hierarchy based on the schematic.

lxSetGroupArrayLocked

```
lxSetGroupArrayLocked(
     d_figId
     g_lockState
     )
     => t / nil
```

Description

Sets the lock attribute on the given group array. If a group array is locked, its X and Y spacing attributes cannot be modified.

Arguments

d_figId	Database ID of the group array that needs to be locked.
g_lockState	A Boolean value indicating whether the group array is set as locked. The default value is `nil`.

Value Returned

`t`	The given group array is locked.
`nil`	The given group array could not be locked.

Example

The following code sets the lock on the group array whose database ID is stored in `groupArray1`.

```
lxSetGroupArrayLocked(groupArray1 t)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxSetGroupArrayRegular

```
lxSetGroupArrayRegular(
     d_figId
     g_regularState
     )
     => t / nil
```

Description

Sets the given group array as regular. For a regular group array, the number of cells can be modified by updating the number of rows or columns. For a non-regular group array, the number of cells is fixed.

Arguments

d_figId	Database ID of the group array.
g_regularState	A Boolean value indicating whether the group array is to be set as regular. The default value is `t`.

Value Returned

`t`	The group array is set as a regular group array.
`nil`	The group array could not be set a regular group array.

Example

The following code sets the group array whose database ID is stored in `groupArray1` as a regular group array.

```
lxSetGroupArrayRegular(groupArray1 t)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxSetNetPinSpecs

```
lxSetNetPinSpecs(
     [ ?nets lt_nets ]
     [ ?lpp lt_lpp ]
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?numPins x_numPins ]
     [ ?create g_create ]
     )
     => t / nil
```

Description

Takes a list of nets and with the specified parameters sets the pin information in either the Generate Layout or Update Components and Nets form. This function must always be called between calls to `lxGenerateStart` and `lxGenerateFinish` or `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

`?nets`	Specifies the net names for which pins are to be generated.
`?lpp`	Layer-purpose pair on which the pins are to be created. The argument is specified using the following syntax. `list("layer" "purpose")` **Note:** If you do not specify this argument, the function defaults to the value returned by the initIOPinLayer environment variable.
`?width`	Specifies the width for each pin. The default is the minWidth value set for the current layer in the technology file. Any change to the value is applied only if the new value is greater than the default value.Pin width in user units.
`?height`	Specifies the height for each pin. The default is the minWidth value set for the current layer in the technology file. Any change to the value is applied only if the new value is greater than the default value.
`?numPins`	Number of pins to create for each of the specified nets.
`?create`	Creates the specified pins.

Value Returned

`t`	Pins options were set.
`nil`	Pins options were not set.

Examples

```
lxGenerateStart(schCv layCv (?extractSchematic t))
  lxSetNetPinSpecs(
    ?nets filteredNets
    ?lpp myLPP
    ?width 2.0
    ?height 2.0)
lxGenerateFinish(schCv layCv (?extractSchematic t))
```

Generates a pin for nets `filteredNets` on layer `myLPP`. Each pin is 2 user units wide and 2 user units high.

```
lxUpdateComponentsAndNetsStart(schCv layCv (?extractSchematic t))
  lxSetNetPinSpecs(
    ?nets filteredNets
    ?lpp myLPP
    ?width 2.0
    ?height 2.0)
lxUpdateComponentsAndNetsFinish(schCv layCv (?extractSchematic t))
```

Updates an existing layout.

lxSetPreserveFloorplanningOptions

```
lxSetPreserveFloorplanningOptions(
     [ ?preserve { all | none } ]
     [ ?rows { t | nil } ]
     [ ?blockages { t | nil } ]
     [ ?areaBoundaries { t | nil } ]
     [ ?trackPatterns { t | nil } ]
     [ ?clusters { t | nil } ]
     [ ?clusterBoundaries { t | nil } ]
     [ ?prBoundary { t | nil}
     [ ?snapBoundary { t | nil } ]
     [ ?snapPatterns { t | nil } ]
     )
     => t / nil
```

Description

Sets the preserve floorplanning objects options in the Generate Layout form. This function must be called between calls to `lxGenerateStart` and `lxGenerateFinish`.

Arguments

`?preserve`	Specifies that either all or none of the floorplanning objects are to be preserved. Enclose the string in quotation marks. The default is `nil`. If you specify this argument, it overrides all other arguments.
`?rows`	Preserves any existing rows and custom placement areas.
`?blockages`	Preserves standalone blockages which have no parent object; a blockage with a parent object is preserved automatically along with its parent. The default is `nil`.
`?areaBoundaries`	Preserves any existing area boundaries. The default is `nil`.
`?trackPatterns`	Preserves any existing track patterns The default is `nil`.
`?clusters`	Preserves any existing clusters. The default is `nil`.
`?clusterBoundaries`	Preserves cluster boundaries, but only if `?clusters` is also set to `t`. The default is `nil`.
`?prBoundary`	Preserves any existing PR boundary. The default is `nil`.
`?snapBoundary`	Preserves the snap boundary but only if `?prBoundary` is also set to `t`. The default is `nil`.

`?snapPatterns`	Preserves snap patterns. The default is `nil`.

Value Returned

`t`	Preserve floorplanning options were set.
`nil`	Preserve floorplanning options were not set.

Examples

Preserves all floorplanning objects except track patterns.

```
lxGenerateStart(schId layId)
  lxSetPreserveFloorplanningOptions(
    ?rows t
    ?boundaries t
    ?areaBoundaries t
    ?clusterBoundaries t
    ?clusters t
    ?trackPatterns nil
    ?prBoundary t
    ?snapBoundary t
    ?snapPatterns t
  )
lxGenerateFinish(schId layId)
```

Preserves all floorplanning objects.

```
lxGenerateStart(schId layId)
  lxSetPreserveFloorplanningOptions(
    ?preserve "all"
  )
lxGenerateFinish(schId layId)
```

lxSetSchematicDriven

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

Description

Sets the various schematic-driven options that are controlled by the *Configure* tab on the Virtuoso Layout Suite XL Connectivity form. The SKILL function can be used to control the following environment variables: `lxSchematicDefaultApp`, `openConnRefTab`, `extractEnabled`, `constraintAwareEditing`, `flightLineEnable`, `autoUpdateFlightlines`, `netNameDisplayEnabled`.

Arguments

g_value	Sets (`t`) or unsets (`nil`) schematic-driven mode for the current session. **Note:** When the schematic-driven mode is unset (`nil`), the `xlStatus` environment variable is set to `nil`. This means the layout Navigator assistant does not display the *XL status* column.

Value Returned

`t`	SKILL function is set.
`nil`	SKILL function is not set.

Additional Information

Depending on the value of the `lxSetSchematicDriven` SKILL function, the `lxSchematicDefaultApp` and `autoUpdateFlightlines` environment variables are set as below:

- When setting to `t`, `lxSchematicDefaultApp` is set to "XL" and `autoUpdateFlightlines` is set to "on".
- When setting to `nil`, `lxSchematicDefaultApp` is set to "None" and `autoUpdateFlightlines` is set to "off".

The `lxSetSchematicDriven` SKILL function controls the value of the `netNameDisplayed` environment variable in each open window. When the SKILL function is called, all the current windows using the environment variable are updated to reflect the value set by the SKILL function. Likewise, the various `extract` environment variables that are controlled by the SKILL function have their values updated for each open layout cellview.

Examples

```
lxSetSchematicDriven(t)
```

Turns the various schematic-driven options.

```
lxSetSchematicDriven(nil)
```

Turns off the various schematic-driven options.

lxSetUpdateOptions

```
lxSetUpdateOptions(
     [ ?deletePins { t | nil } ]
     [ ?deleteInstances { t | nil } ]
     [ ?replaceMaster { t | nil } ]
     [ ?withMarkers { t | nil } ]
     [ ?netsOnly { t | nil } ]
     [ ?selected nil { t | nil } ]
     [ ?layoutParameters { t | nil } ]
     [ ?layoutConstraints { t | nil } ]
     [ ?sigType { t | nil } ]
     [ ?netVoltage { t | nil } ]
     )
     => t / nil
```

Description

Sets the options in the *Update* group box of the Update Components and Nets form. It must be called between calls to `lxUpdateComponentsAndNetsStart` and `lxUpdateComponentsAndNetsFinish`.

Arguments

`?deletePins`	Deletes layout pins that are no longer present in the schematic. Redundant nets and terminals are deleted from the layout view at the same time.
`?deleteInstances`	Deletes layout instances that are no longer present in the schematic.
`?replaceMaster`	Updates any existing instances that use an incorrect master with instances that use the correct master. Use together with `?withMarkers` to specify how the new instances are added to the design.
`?withMarkers`	When set to `t`, the system puts a marker on the incorrect instance in the layout canvas and renames it *`name`*`_old`. It then creates a new instance with the correct master and places it below the design boundary. When set to `nil`, the system updates the instance in place to use the correct master. **Note:** This option is honored only when `?replaceMaster` is set to `t`.
`?netsOnly`	Updates only net assignments and instance, terminal, and net names. Enabling this option also automatically preserves user-defined bindings. All other options are ignored, except `?selected`.
`?selected`	Updates only the instances and pins currently selected in the layout window.
`?layoutParameters`	Updates the parameters and parameter values on layout instances to match those on their schematic counterparts. Parameters that are set in layout instances but are not present on their schematic counterparts are not removed.
`?layoutConstraints`	Updates the layout constraints to match their schematic counterparts.
`?sigType`	Updates the layout net signal types to match their schematic counterparts.

`?netVoltage`	Updates the minimum and maximum voltages on layout nets to match their schematic counterparts.

Value Returned

`t`	Update options were set.
`nil`	Update options were not set.

Example

```
lxUpdateComponentsAndNetsStart(schId layId (?extractSchematic t))
  lxSetUpdateOptions(
    ?deletePins t
    ?deleteInstances t
    ?replaceMaster t
    ?withMarkers nil
    ?netsOnly nil
    ?selected nil
    ?layoutParameters nil)
    ?layoutConstraints nil)
lxUpdateComponentsAndNetsFinish(schId layId (?extractSchematic t))
```

Removes unmatched pins and instances from the layout view and replaces any instances with incorrect masters with new instances of the correct master.

lxShapeSlotting

```
lxShapeSlotting(
     [ ?cv d_cellViewId ]
     [ ?all { t | nil } ]
     [ ?region l_region ]
     [ ?layers l_layers
     [ ?widthThreshold n_widthThreshold ]
     [ ?slotLength n_slotLength ]
     [ ?slotWidth n_slotWidth ]
     [ ?lSpacing n_lSpacing ]
     [ ?wSpacing n_wSpacing ]
     [ ?slotToEdge n_slotToEdge ]
     [ ?slotVia { t | nil } ]
     [ ?windowSize n_windowSize ]
     [ ?stepSize n_stepSize ]
     [ ?maxDensity n_maxDensity ]
     [ ?slotStaggered { t | nil } ]
     [ ?lengthWidthRatio n_lengthWidthRatio ]
     [ ?excludePins { t | nil } ]
     [ ?slotSpacingAtTurn n_slotSpacingAtTurn ]
     [ ?onlyComplexPolygons { t | nil }
     [ ?halfManhattan { t | nil } ]
     [ ?halfManhattanStyle l_halfManhattanStyle ]
     [ ?directionalSlotPattern l_directionalSlotPattern ]
     [ ?shapes l_shapes ]
     [ ?turnSlotPattern l_turnSlotPattern ]
     )
     => t / nil
```

Description

Creates slots on shapes to ensure that wide wires are in compliance with `maxDensity` and `maxWidth` constraint rules. It also preserves viable electromigration and resistance attributes by creating rectangular slots in the direction of the current over the wires and square slots at turns (same layer or changed layer). There are two modes depending on the arguments specified: Geometric and Density Aware. In Geometric mode, the slotting shapes are fully specified (size, spacing, and step size), while in Density Aware mode, the slotting shape geometries are automatically computed based on the `maxDensity` specification. Also, in Geometric mode, you can create slots on shapes on non-metal layers. It is still possible to override any of the geometric information, such as the width, length, or spacing.

Important

The `lxShapeSlotting` SKILL API is currently supported for Linux platform only.

Arguments

`?cv`	ID of the cellview containing the layout instance. If not specified, the current cellview is used.
`?all`	Applies the command to the entire design when set to `t`. Default is the entire design.
`?region`	Applies the command to the region given by a list of coordinates (x0 y0 x1 y1) representing the lower-left and upper-right corners of the region.
`?layers`	List of layer names on which the `drawing` purpose metal shapes will be overlapped by `slot` purpose shapes. The default is all layers if none is specified.
`?widthThreshold`	Filters out the shapes with a width smaller than the one specified. Default is `12.0µm`.
`?slotLength`	Defines the length of the slot. Default is `0.0µm`.
`?slotWidth`	Defines the width of the slot. Default is `0.0µm`.
`?lSpacing`	Defines the slot spacing in the length direction. Default is `0.0µm`.
`?wSpacing`	Defines the slot spacing in the width direction. Default is `0.0µm`.
`?slotToEdge`	Defines the minimum slot to edge distance. Default is `0.0µm`.
`?slotVia`	When set to `t`, vias are slotted. Default is `nil`.
`?windowSize`	Defines the size of the window where density is measured. Default is `20µm`.
`?stepSize`	Defines the step applied to the window for the next check. It is usually half of the window. Default is `10 µm`.
`?maxDensity`	Specifies the maximum metal density that is allowed for the layer as a percentage. Default is 80 percent.
`?slotStaggered`	Defines the offset mode used to add the slots. If the offset mode is set to `nil`, slots are inline. However, if the offset mode is set to `t`, slots are staggered. By default the offset mode is set to `nil`.

`?lengthWidthRatio` Specifies a threshold ratio value which determines whether a slot is created as a square or a rectangular shape. If the length-to-width ratio of the object is greater than the defined threshold, then a rectangular slot is created. If the length-to-width ratio of the object is less than or equal to the specified ratio, then a square slot is created. The default is `3.0`.

`?excludePins` Excludes pin objects from being slotted. Default is `nil`.

`?slotSpacingAtTurn`

Specifies the spacing that needs to be applied at the turns or on vias.

If this argument is not specified:

- In geometric mode, the slot shapes at turns are aligned with the slot shapes from the wires.
- In density-aware mode, the command computes the spacing between the slots in the turn from the target density.

When the argument is specified:

- If the value = 0, then the slots in the turn are aligned with the slots of the shapes touching the turn.
- If the value > 0, then the value is used as the spacing between the slots in the turn.

`?onlyComplexPolygons`

Slots only complex polygons. The bounding box of the polygon is used to determine the direction of the current. The shape of the slot depends on the `?lengthWidthRatio` value.

`?halfManhattan` Enables generation of 45 degrees slots over 45 degree wires.

`?halfManhattanStyle` Specifies the type of slot shape to be dropped at the turn when `halfMahattan` is set to `t`. The slot shape can be either a `square`, `bentShape` or `rowExtension`. Default is `square`.

`?directionalSlotPattern`

Defines a slot pattern (for example, a complex chamfered rectangle) that is dropped in the direction of the current over the orthogonal portion of the wire.

`?shapes`	List of shapes to be slotted. The shape can be a path segment, path, rectangle, or a polygon. If `?region` is also specified, then only the section of the shapes that are partially or fully covered by the `?region` are slotted.
`?turnSlotPattern`	Defines a slot pattern (for example, a complex chamfered square) that is dropped at the turn of a wire.

Value Returned

`t`	Slots are created on metal shapes.
`nil`	Fails to create slots on metal shapes.

Example

```
lxShapeSlotting(?windowSize 20.0 ?stepSize 10.0 ?lengthWidthRatio 0.5 ?maxDensity
80.0 ?widthThreshold 10.0 ?all ?layers list("Metal2" "Metal1") ?slotVia t
?slotPattern 0 ?slotLength 5.0 ?slotWidth 1.0 ?lSpacing 2.0 ?slotStaggered t)
```

Creates rectangular slot shapes with geometries specified as `5.0µm` in length and `1.0µm` in width spaced by `2.0µm` over `Metal1` and `Metal2` wires wider than `10.0µm`. This happens where `maxDensity` violations exist. Also, in this situation, the spacing in the width direction is computed automatically (corresponding to the `?wSpacing` parameter).

lxShowCommonIncompleteNetsForSelectedInsts

```
lxShowCommonIncompleteNetsForSelectedInsts(
     [ w_windowID ]
     )
     => t / nil
```

Description

Runs the Enter function named *Show Common Incomplete Nets for Selected Instances*, which displays the incomplete nets that are common to the instances selected in the layout. The SKILL function can also be run in pre-select mode on the set of instances that are already selected in the layout. The post-selection mode is supported when only one instance is pre-selected.

Arguments

w_windowID	An optional layout window ID in which you want the Enter function to run. If you do not specify a window ID, the current window is used.

Value Returned

`t`	The Enter function was run.
`nil`	The Enter function failed to run.

Example

```
lxShowCommonIncompleteNetsForSelectedInsts()
```

lxShowHideIncompleteNets

```
lxShowHideIncompleteNets(
     [ w_windowID ]
     )
     => t / nil
```

Description

Shows and hides any incomplete nets associated with selected objects in the specified layout window. The function establishes the incomplete nets associated with the selected objects and toggles the visibility of the markers representing those incomplete nets on or off depending on their previous state.

You can select objects either before or after you run the function, which behaves differently in each case.

- In *preselection mode*, you select some objects in the canvas and then type the command in the CIW. Any incomplete nets associated with the selected objects are displayed in the canvas.

  Note: In this mode, the function runs only once on the set of selected objects. You must re-type it each time you want to run it for a new set of objects.

- In *postselection mode*, you type the function name in the CIW with nothing selected and then select objects in the design canvas. Any incomplete nets associated with the selected objects are displayed. To hide the incomplete nets associated with an object, click that object again. When you have multiple objects selected, use `Ctrl`-click to remove an individual object from the selected set and hide its incomplete nets.

  Note: In this mode the function remains active until you cancel it by pressing either `Esc` or `Ctrl-C` on the keyboard. It is canceled automatically when you start another enter function.

To be able to show flight lines for open markers, the Annotation Browser must be initialized, although it need not be displayed on the desktop. For example, you can close it and show and hide incomplete nets in postselection mode directly from the Navigator assistant. To do this, though, you must switch off the *Remove highlighting when browser is closed* option in the Annotation Browser Options form, otherwise there can be no visible incomplete nets to show and hide.

Arguments

w_windowID	ID of the layout window in which you want the function to operate. If you do not specify a window ID, the current window is used.

Value Returned

t	The command was launched.
nil	The command failed to launch.

lxToggleLocalAbutment

```
lxToggleLocalAbutment(
     )
     => t
```

Description

Changes the state of the `lxLocalAbutment` environment variable and honors the change immediately in Layout XL. Use this function to switch local abutment on and off without having to restart the software.

Tip

If you change the lxLocalAbutment environment variable in the CIW, the change takes effect only after you restart Layout XL.

To be able to show flight lines for open markers, the Annotation Browser must be initialized, although it need not be displayed on the desktop. To show markers when the browser is hidden, you must switch off the *Remove highlighting when browser is closed* option in the Annotation Browser Options form, otherwise there can be no visible incomplete nets to show and hide.

Arguments

None

Value Returned

`t`	The local abutment functionality was toggled successfully.

lxToggleShowAllIncompleteNets

```
lxToggleShowAllIncompleteNets(
     [ w_windowID ]
     )
     => t / nil
```

Description

Toggles the visibility of all the incomplete nets displayed in the design canvas.

Arguments

w_windowID	ID of the layout window in which you want the function to operate. If you do not specify a window ID, the current window is used.

Value Returned

t	The command was successful.
nil	The command failed.

lxToggleShowIncompleteNets

```
lxToggleShowIncompleteNets(
     [ w_windowID ]
     )
     => t / nil
```

Description

Toggles the visibility of the current set of incomplete nets displayed in the design canvas. The first time you run the command, it hides all currently visible incomplete nets and stores the list of incomplete nets for later retrieval. The next time you run the command, it restores the visibility of the stored set of incomplete nets.

Important

If you run the function for the first time on a design with no incomplete nets on display, it does nothing because there is no stored list of nets to display.

To be able to show incomplete nets, the Annotation Browser must be initialized. It need not be displayed on the desktop; however, if you do hide it, you must switch off the *Remove highlighting when browser is closed* option in the Annotation Browser Options form, otherwise there can be no visible incomplete nets to show and hide.

Arguments

w_windowID	ID of the layout window in which you want the function to operate. If you do not specify a window ID, the current window is used.

Value Returned

t	The command was successful
nil	The command failed.

lxUnfold

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

Description

Deletes all the folds of the selected folded device and creates a single, unfolded device at the same coordinates in the layout as the selected fold. The width of the new unfolded device is determined based on the width of the corresponding schematic device, if one exists. If the corresponding schematic device does not exist and the `unfoldUseLayoutWidths` environment variable is set to `t`, the sum of the fold widths is used to determine the total width of the unfolded device.

Arguments

None

Value Returned

`t`	The folds of the selected layout device are deleted, and an unfolded device is created.
`nil`	The command failed possibly because multiple folded devices were selected.

Examples

```
cv = geGetEditCellView()
p0 = dbFindAnyInstByName(cv "P0")
lxFold(list(p0) 2:2 3)
```

Creates three folds (`P0.1`, `P0.2` and `P0.3`) starting at the coordinate `2:2`.

```
p01 = dbFindAnyInstByName(cv "P0.1")
geSelectFig(p01)
lxUnfold()
```

Deletes `P0.1`, `P0.2` and `P0.3`, and creates `P0` at the coordinate `2:2`.

lxUnregMasterDiffName

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

Description

Unregisters the SKILL function used to return a name for a master difference reported by the *Check Against Source* command.

Argument

None

Value Returned

`t`	The SKILL function was unregistered.
`nil`	The SKILL function was not unregistered.

Example

```
lxUnregMasterDiffName()
```

lxUnregPrePosition

```
lxUnregPrePosition(
     t_lib
     t_cell
     t_view
     )
=> t / nil
```

Description

Unregisters the SKILL function previously registered using `lxRefPrePosition` for the specified schematic Pcell.

Arguments

t_lib	Name of a library.
t_cell	Name of a cell.
t_view	Name of a view for the schematic Pcell.

Value Returned

`t`	The function is unregistered.
`nil`	The function is not unregistered.

Example

Unegisters the pre-position SKILL function for a schematic Pcell with library `schLib`, cell name `inv`, and view name `schematic`.

```
lxUnregPrePosition("schLib" "inv" "schematic")
```

Use the following to call a registered function with the cellview.

```
lxRegPrePosition("" "" "" 'preFunc)
```

Related Topics

lxRegPrePosition

lxUnSlotVia

```
lxUnSlotVia(
     [ ?cv d_cellViewId ]
     [ ?all { t | nil } ]
     [ ?region l_region ]
     [ ?layers l_layers ]
     )
     => t / nil
```

Description

Restores the via that has been slotted by the command lxShapeSlotting.

Arguments

`?cv`	ID of the cellview containing the layout instance. If not specified, the current cellview is used.
`?all`	Applies the command to the entire design when set to `t`. Default is the entire design.
`?region`	Applies the command to the region given by a list of coordinates (x0 y0 x1 y1), which represent the lower-left and upper-right corners of the region.
`?layers`	List of layer names on which the slotted vias are restored to original. This affects via arrays that contain at least one pair of adjacent layers. **Note:** Unslotting of vias does not take place on a single layer and if the layer pair does not consists of adjacent layers, such as `Metal2` and `Metal5`.

Value Returned

`t`	When the arguments are valid.
`nil`	When the arguments are invalid.

Examples

Example1

```
lxUnSlotVia(?all t)
```

Unslots all the vias in the cellview being edited.

Example2

```
lxUnSlotVia(?all ?layers list("Metal1" "Metal2"))
```

Unslots the `Metal1-Metal2` vias.

lxUpdateAllPhysBinding

```
lxUpdateAllPhysBinding(
     d_layCV
     )
     => t / nil
```

Description

Updates the physical bindings of all the instances in the specified layout cellview.

Arguments

d_layCV	The database ID of the layout cellview for which all instance bindings are to be updated.

Value Returned

`t`	The binding for at least one instance was successfully updated.
`nil`	No bindings were updated.

Example

You have modified the masters of two instances in the layout. Calling this function updates the masters for both of these instances and the update is reflected in the physConfig view.

lxUpdateBinding

```
lxUpdateBinding(
     d_layCV
     [ ?schCV d_schCV ]
     [ ?nameOnly { t | nil } ]
     [ ?currentLevel { t | nil } ]
     [ ?extract { t | nil } ]
     [ ?clearConn { t | nil } ]
     [ ?extractStopLevel [0-32] ]
     [ ?preserveUserBindings { t | nil } ]
     [ ?preserveLayoutHierarchy { t | nil } ]
     [ ?flattenLayoutCreate { Synchronized Family | Grouped Objects| Free Objects
     } ]
     [ ?bindingFile t_fileName ]
     [ ?crossRefFile t_fileName ]
     [ ?extractedNetlistFile t_fileName ]
     [ ?PVSRulesFile t_fileName ]
     [ ?PVSScale n_PVSScale ]
     [ ?correctMaster { t | nil } ]
     [ ?ignoreRouteCells { t | nil } ]
     [ ?ignoreDummies { t | nil } ]
     [ ?preserveExistingBindings { t | nil } ]
     [ ?hierarchical { t | nil } ]
     [ ?allCellsInDesign { t | nil } ]
     [ ?createTransparent { t | nil } ]
     [ ?createPinsFromLabels { t | nil } ]
     [ ?checkDuplicateObjects { t | nil } ]
     [ ?removeDuplicateObjects { t | nil } ]
     [ ?readOnlyLayout { t | nil } ]
     )
     => 0 - 100 / -1
```

Description

Improves the binding between the schematic and the layout by setting options that control how the connectivity, the manual bindings, and the design hierarchy is processed. If a log file name is specified for the hierSummaryLogFileName environment variable, the SKILL function prints the summary information to the specified log file, in addition to printing the information to the default Virtuoso log file

Arguments

d_layCV

The database ID of the layout cellview for which the bindings are to be improved.

d_schCV

The database ID of the schematic cellview for which the bindings are to be improved.

?nameOnly

Binds by name, if true.

The default is `nil`, which means instances are bound based on connectivity.

?currentLevel

Binds at the current level, if true.

The default is `t`.

If set to `nil`, the schematic and layout hierarchies are flattened to find leaf-level bindings. The *Update Binding* command is run hierarchically for the current layout.

?extract

Performs physical connectivity extraction to ensure that the layout connectivity is up-to-date.

The default is `t`.

Note: The extraction can only be disabled if `currentLevel`is set to `t` and `clearConn` is set to `nil`.

?clearConn

Removes the logical connectivity in the layout cellview.

The default is `nil`.

?extractStopLevel

Specifies the hierarchy depth for extraction.

The default is `0`.

The extraction depth must be an integer between `0` and `32`.

`?preserveUserBindings`

Preserves the user-defined bindings created by using *Define Device Correspondence*.

The default is `t`.

If set to `nil`, all the user-defined bindings will be deleted.

`?preserveLayoutHierarchy`

If set to `t`, the layout hierarchy is preserved and complex bindings are created for hierarchy mismatches between the schematic and the layout.

Note: If `createTransparent` is set to `t`, the `preserveLayoutHierarchy` argument can set instances to be transparent instead of creating complex bindings.

If `preserveLayoutHierarchy` is set to `nil`, layout instances are flattened to match the schematic hierarchy.

The default is `nil`.

`?flattenLayoutCreate`

If the layout hierarchy is not preserved, optionally creates *fig groups* or *synchronous clones* for flattened layout instances.

The default is "`Synchronized Family`".

Note: The value that you specify for this argument should be enclosed in quotation marks, such a**s** ("`Synchronized Family`" | "`Grouped Objects`" | "`Free Objects`").

`?bindingFile` *`t_fileName`*

Path to the binding file that is created by the `lxCreateBndFile` SKILL function.

`?crossRefFile` *`t_fileName`*

Path to the PVS instance cross-reference file to be used for creating the binding file.

Note: PVS creates a `.ixf` cross-reference file in the `svdb` directory of the PVS run directory when using the PVS **Create QRC Input Data** option.

`?extractedNetlistFile` *`t_fileName`*

Path to the extracted layout CDL netlist file to be used for creating the binding file.

Note: PVS creates a `.net` extracted netlist file in the `svdb` directory of the PVS run directory when using the PVS **Create QRC Input Data** option. If a `.net` file is unavailable, the binding file can be created using a `.spi` file.

`?PVSRulesFile` *`t_fileName`*

Path to the PVS rule file to be used for creating the binding file if the scale at which PVS was run is different from the layout scale.

Note: If the scale at which PVS was run is the same as the layout scale, the binding file can be created, as usual, based only on the instance cross reference file and the extracted netlist file.

`?PVSScale` *`n_PVSScale`*

Specifies the floating point input scale for running PVS.

By default, the input_scale is `0`, which means that `dbuPerUu` for the layout is used. But, if the PVS input scale is different from that of the layout, and the PVS rule file is specified, the PVS rule deck is parsed to determine the appropriate value. Alternatively, you can specify the PVS input scale to be used.

`?correctMaster`

Binds a schematic and layout instance if the layout instance has a master that corresponds to the physical binding defined in the CPH window.

If the current physical binding does not match the bound layout instance, the SKILL function updates the physical bindings in CPH. The physical bindings in CPH can only be updated when the bindings are not already in use.

The default is `t`.

`?ignoreRouteCells`

Adds an `ignore` property on layout instances that are identified as route cells to ignore the instances for binding.

Route cells are identified as cells that have no devices using extract stop layers. For cases when no extract stop layers are defined, route cells are identified as the ones that have no terminals but only metal or cut shapes.

`?ignoreDummies`

Ignores any unbound, potential dummy instances that are identified during a binding run.

`?preserveExistingBindings`

Preserves not only the user bindings but also any existing bindings, if set to `t`.

The default is `nil`.

If set to `nil`, all previous bindings, including any user-defined bindings previously added, will be deleted.

Note: For the user-defined bindings to be deleted when `preserveExistingBindings` is set to `nil`, `preserveUserBindings` must also be set to `nil`.

`?hierarchical`

Runs the Update Binding command at current level for each layout cellview in the current layout design hierarchy.

The default value is `nil`.

When set to `t`:

- The *Update Binding* command is run at the current level for each layout and schematic cellview pair found in the design hierarchy. At the end of the command, the XL status is displayed for each layout cellview with connectivity reference, and a list of layout cellviews with no connectivity reference is also displayed.

  The XL status provides information on:

  - Number of bound, unbound, and ungenerated instances
  - Master, parameter, and connectivity differences
  - List of cellviews identified as leafs

- The *Update Binding* command prints the summary information to the specified log file, in addition to printing the information to the default Virtuoso log file, when a log file name is specified for the hierSummaryLogFileName environment variable.

`?allCellsInDesign`

Binds all non-leaf cellviews at current level by using their corresponding connectivity reference.

The default is `nil`.

Note: This argument provides the same functionality as `?hierarchical`, which is being supported to allow backward compatibility.

`?createTransparent`

Preserves the layout hierarchy and sets instances as transparent, so that bindings between the leaf-level instances within the instance master and their schematic instances can be created.

`?createPinsFromLabels`

Creates pins from labels, if the label text or text display matches an ungenerated schematic terminal name. createPinsFromLabels

`?checkDuplicateObjects`

Checks and reports duplicate instances, shapes, and vias in the layout cellview.

`?removeDuplicateObjects`

Checks and removes duplicate instances, shapes, and vias from the layout cellview.

`?readOnlyLayout` When set to `t`, runs on a read-only layout without DM system prompting the user to change from read-only to edit mode. The default is `nil`.

Value Returned

`0-100`	The average compliance.
`nil`	The check failed.

Example 1

```
lxUpdateBinding(layCV ?schCV schCV ?clearConn t ?preserveUserBindings nil
?preserveExistingBindings nil ?preserveLayoutHierarchy nil ?flattenLayoutCreate
"Group" ?bindingFile "./inv.bnd" ?hierarchical nil)
```

Updates the binding between the schematic and layout cellviews using the binding file "`./inv.bnd`", Deletes the logical connectivity and user bindings, and flattens the layout hierarchy. For any flattened layout instances, adds the flattened instances to a figGroup.

```
lxUpdateBinding(layCV ?schCV schCV ?preserveLayoutHierarchy t ?crossRefFile "./
inv.ixf" ?extractedNetlistFile "./inv.net)
```

Updates the binding between the schematic and layout cellviews using the PVS instance cross-reference file "`./inv.inf`" and the extracted layout CDL netlist file "`inv.net`", preserving the logical connectivity, user bindings, and the layout hierarchy.

lxUpdateComponentsAndNets

```
lxUpdateComponentsAndNets(
     d_schCellViewID
     d_layCellViewID
     [ ?initCreatePins { t | nil } ]
     [ ?initGlobalNetPins { t | nil } ]
     [ ?initCreatePadPins { t | nil } ]
     [ ?initCreateInstances { t | nil } ]
     [ ?initCreateBoundary { t | nil } ]
     [ ?initCreateSnapBoundary { t | nil } ]
     [ ?initDoStacking { t | nil } ]
     [ ?initDoFolding { t | nil } ]
     [ ?initCreateMTM { t | nil } ]
     [ ?deleteUnmatchedInsts { t | nil } ]
     [ ?deleteUnmatchedPins { t | nil } ]
     [ ?updateReplacesMasters { t | nil } ]
     [ ?updateWithMarkers { t | nil } ]
     [ ?updateLayoutParameters { t | nil } ]
     [ ?updateNetSigType { t | nil } ]
     [ ?updateNetMinMaxVoltage { t | nil } ]
     [ ?updateNetsOnly { t | nil } ]
     [ ?virtualHierarchy { t | nil } ]
     [ ?extractSchematic { t | nil } ]
     )
     => t / nil
```

Description

Updates the components and nets for the specified schematic and layout cellviews. If you use the optional arguments, they override the existing environment variables set in the `.cdsenv` file. The default value for each optional argument corresponds to the default value of the associated environment variable.

Arguments

d_schCellViewID

Database ID of the source schematic cellview.

d_layCellViewID

Database ID of the layout cellview to be updated.

?initCreatePins

Generates the pins in the design and snaps them to the placement grid.

Environment variable: initCreatePins

?initGlobalNetPins

Generates layout pins for the global nets in the schematic.

Note: This option is honored only when ?initCreatePins is set to t.

Environment variable: initGlobalNetPins

?initCreatePadPins

Generates layout pins and pads for schematic pins that are connected to I/O pads.

Note: This option is honored only when ?initCreatePins is set to t.

Environment variable: initCreatePadPins

?initCreateInstances

Generates all the instances in the schematic that do not have one of the ignore properties attached to them.

Environment variable: initCreateInstances

?initCreateBoundary

Generates a place and route boundary based on the settings you make in the *Boundary* tab of the Generate Layout form.

Environment variable: initCreateBoundary

?initCreateSnapBoundary

Generates a rectangular snap boundary that encloses the generated place and route boundary.

Note: This option is honored only when `?initCreateBoundary` is set to `t`.

Environment variable: `initCreateSnapBoundary`

`?initDoStacking`

Automatically abuts MOS transistors into chains during layout generation.

Environment variable: `initDoStacking`

`?initDoFolding`

Automatically splits devices into fingers to prevent gate width from exceeding a specific size.

Environment variable: `initDoFolding`

`?initCreateMTM`

Preserves user-defined one-to-one, many-to-many, many-to-one, and one-to-many device correspondence defined in the Define Device Correspondence form. It does not preserve user-specified internal bindings made using the Choose Binding form; nor does it report missing devices or shapes within a bound group.

Environment variable: `initCreateMTM`

`?deleteUnmatchedInsts`

Deletes layout instances that are no longer present in the schematic and creates markers in the layout view.

Environment variable: `deleteUnmatchedInsts`

`?deleteUnmatchedPins`

Deletes layout pins that are no longer present in the schematic. Redundant nets and terminals are deleted from the layout view at the same time.

Environment variable: `deleteUnmatchedPins`

`?updateReplacesMasters`

Updates any existing instances that use an incorrect master with instances that use the correct master.

Use together with `?updateWithMarkers` to specify how the new instances are added to the design.

Environment variable: `updateReplacesMasters`

`?updateWithMarkers`

When set to `t`, the system puts a marker on the incorrect instance in the layout canvas and renames it *`name`*`_old`. It then creates a new instance with the correct master and places it below the design boundary.

When set to `nil`, the system updates the instance in place to use the correct master.

Note: This option is honored only when `?updateReplacesMasters` is set to `t`.

Environment variable: `updateWithMarkers`

`?updateLayoutParameters`

Updates the parameters and parameter values on layout instances to match those on their schematic counterparts. Parameters that are set in layout instances but are not present on their schematic counterparts are not removed.

Environment variable: `updateLayoutParameters`

`?updateNetSigType`

Updates the layout net signal types to match their schematic counterparts.

Environment variable: `updateNetSigType`

`?updateNetMinMaxVoltage`

Updates the minimum and maximum voltages on layout nets to match their schematic counterparts.

Environment variable: `updateNetMinMaxVoltage`

`?updateNetsOnly`

	Updates only net assignments and instance, terminal, and net names. Enabling this option also automatically preserves user-defined bindings. All other options are ignored. Environment variable: `updateNetsOnly`
`?virtualHierarchy`	
	Generates virtual hierarchy for schematic symbols that do not have any corresponding layouts. Environment variable: `generateVirtualHierarchy`
`?extractSchematic`	
	Extracts the schematic design, if required. If the schematic needs to be extracted and this option is not set, the system issues a message and layout generation stops. Environment variable: `initCreateSnapBoundary`

Value Returned

`t`	Update components and nets was successful.
`nil`	Update components and nets failed.

Example

```
schCV=dbOpenCellViewByType( schLibName schCellName schViewName )
layCV=dbOpenCellViewByType( layLibName layCellName layViewName )
lxUpdateComponentsAndNets(schCV layCV)
```

Updates the specified layout to match the specified schematic using the default settings from the `.cdsenv`.

Note: You must perform an explicit `dbSave` after you have updated the layout cellview.

lxUpdateComponentsAndNetsFinish

```
lxUpdateComponentsAndNetsFinish(
     d_schId
     d_layId
     [ ?extractSchematic { t | nil } ]
     )
     => t / nil
```

Description

Finishes the update process by executing the *Update Components And Nets* command with the form values supplied by `lxSet*` functions and after the previous call to `lxUpdateComponentsAndNetsStart`.

When used with the other `lxSet*` functions, this provides an alternative interface to the Update Components And Nets form. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

d_schId	Database ID of the schematic cellview that was used as the connectivity source.
d_layId	Database ID of the layout cellview in which components are to be generated and updated.
?extractSchematic	Automatically extracts the schematic if required. If the schematic needs to be extracted and this option is not set, the system issues a message and the update stops.

Value Returned

t	Layout update completed.
nil	Layout update did not complete. Check the log file for details.

Example

```
lxUpdateComponentsAndNetsStart(schCv layCv (?extractSchematic t))
...
;; lxSet* calls used to set form values
```

```
....
lxUpdateComponentsAndNetsFinish(schCv layCv (?extractSchematic t))
```

lxUpdateComponentsAndNetsStart

```
lxUpdateComponentsAndNetsStart(
     d_schId
     d_layId
     [ ?extractSchematic { t | nil } ]
     )
     => t / nil
```

Description

Starts the *Update Components And Nets* command, building the Update Components and Nets form without displaying it to the screen.

When used with the `lxGet*` and `lxSet*` functions, this provides an alternative interface to the Update Components and Nets form. If optional arguments are not supplied the normal default values for the form fields apply.

Arguments

d_schId	Database ID of the schematic cellview to be used as the connectivity source.
d_layId	Database ID of the layout cellview in which components will be generated and updated.
?extractSchematic	Automatically extracts the schematic if required. If the schematic needs to be extracted and this option is not set, the system issues a message and the update stops.

Value Returned

t	The Update Components and Nets form was initialized.
nil	The Update Components and Nets form was not initialized. Check that you have a Layout XL license available, and then check the log file for details.

Example

```
hiSetCurrentWindow(schWin)
schCv = geGetEditCellView()
```

```
hiSetCurrentWindow(layWin)
layCv = geGetEditCellView()
....

 lxUpdateComponentsAndNetsStart(schCv layCv (?extractSchematic t))
```

Opens the Update Components and Nets form for the current views in the schematic and layout windows and extracts the schematic, if required.

lxUpdateGroupArrayParams

```
lxUpdateGroupArrayParams(
     d_groupArrayId
     [ ?rows x_rows ]
     [ ?columns x_cols ]
     [ ?X x_xValue ]
     [ ?Y x_yValue ]
     [ ?orients l_orientList ]
     [ ?mode 'pitch | 'spacing  ]
     [ ?lpp l_refLPP ]
     )
     => t / nil
```

Description

Updates the specified parameters of a group array.

Arguments

d_groupArrayId	Database ID of the group array to be updated.
?rows *x_rows*	Number of rows in the group array.
?columns *x_cols*	Number of columns in the group array.
?X *x_xValue*	Spacing added between group array cells in the X direction. Spacing is applied based on the specified mode.
?Y *x_yValue*	Spacing added between group array cells in the Y direction. Spacing is applied based on the specified mode.
?orients *l_orientList*	Orientation pattern used for the group array. If the pattern is smaller than the given rows and columns in the group array, the pattern is repeated from left to right and bottom to top. The bottom-most row in the pattern is considered as the first row.
?mode 'pitch | 'spacing	

	Mode used for applying spacing between group array cells. Valid values are `'pitch` and `'spacing`. ■ `'pitch`: Lets you specify the distance that is measured between the same-edge corners of cells. For example, the distance between the left edges of two cell. ■ `'spacing`: Lets you specify the distance that is measured between the adjacent or closest edges of cells. For example, the distance between the top edge of a cell and the lower edge of another cell.
`?lpp` *`l_refLPP`*	Specifies the layer-purpose pair used for computing the bounding box of the cells in a group array. The combined bounding box of the geometries on the specified reference layer-purpose pair is used for calculating the spacing between cells.

Value Returned

`t`	Group array parameters updated successfully.
`nil`	Group array parameters could not be updated.

Example

The following code updates the row and column count and the X and Y spacing of the group array whose database ID is stored in `groupArray1`.

```
lxUpdateGroupArrayParams(groupArray1 ?rows 3 ?columns 3 ?X 4 ?Y 3)
==> t
```

Related Topics

Group Arrays

Group Array SKILL Functions

lxUpdatePhysBinding

```
lxUpdatePhysBinding(
     d_layCV
     )
     => t / nil
```

Description

Updates the physical bindings of the currently selected instances in the specified layout cellview.

Arguments

d_layCV	The database ID of the layout cellview for which the bindings of currently selected instances are to be updated.

Value Returned

`t`	The binding for at least one instance was successfully updated.
`nil`	No bindings were updated.

Example

You have modified the masters of two instances in the layout and one of these instances is selected. Calling this function updates the master for the selected instance only. The update is reflected in the physConfig view.

lxUpdatePlacementStatus

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

Description

Updates the placement status of instances and pins in the current cellview. Instances and pins that are wholly inside the PR boundary with status `unknown` are updated to status `placed`. Instances and pins wholly outside the PR boundary with status `placed` are updated to `unknown`.

Arguments

None

Value Returned

`t`	The placement status was updated.
`nil`	The placement status was not updated.

Example

```
lxUpdatePlacementStatus()
```

Additional Information

Use this function to update the placement status of instances or pins in designs that are loaded in Layout XL for the first time, or which have been edited outside the Layout XL environment.

You can set the updatePlacementStatus environment variable to `t` to update the placement status automatically whenever an instance or pin is moved into or out of the PR boundary (including situations where the PR boundary is moved or stretched to enclose or exclude an instance or pin).

lxUpdateSchematicParameters

```
lxUpdateSchematicParameters(
     d_schematicId
     d_layoutId
     )
     => t / nil
```

Description

Updates the schematic parameters for all the layout instances bound to the top-level schematic instances.

Arguments

d_schematicId	Database ID of a schematic cellview.
d_layoutId	Database ID of a layout cellview.

Value Returned

`t`	Updates the schematic parameters for all the layout instances bound to the top-level schematic instances.
`nil`	The operation failed.

Examples

Updates parameter differences for top-level schematic instances bound to layout instances in the specified design `hierarchy`, `ampString`, or `layout`.

```
scv = dbOpenCellViewByType("hierarchy" "ampString" "schematic" "schematic" "r")
lcv = dbOpenCellViewByType("hierarchy" "ampString" "layout" "maskLayout" "r")

lxUpdateSchematicParameters(scv lcv)
```

Related Topics

Schematic Parameter Updates

lxVerifyCloneFamily

```
lxVerifyCloneFamily(
     [ cloneList ]
     )
     => t / nil
```

Description

Runs a consistency check on the specified synchronous clone family to verify if the family members are identical to each other.

Argument.

cloneList	The cloneList can be: ■ A single synchronous clone ■ A list of synchronous clones ■ Nothing specified **Note:** ■ When the function is run with no argument specified, the function is called on selected figures, such as, when calling `lxVerifyCloneFamily(geGetSelSet())` ■ If the argument list contains a non-synchronous clone object, the object is ignored.

Value Returned

`t`	The Consistency Check has run successfully on all the identified clone families.
`nil`	The value returned is `nil` if: ■ Objects specified are non-synchronous clones or when the function is run with no arguments specified and the selection set has no synchronous clones. ■ The Consistency Check has failed because the clone family members are not identical to each other. ❑ In this case, a warning message is displayed, indicating that the clone family name has been removed during the Consistency Check because the clone members were not found to be identical.

Example

```
lxVerifyCloneFamily(cloneList)
```

nclRunConstraintValidation

```
nclRunConstraintValidation(
     d_cellviewID
     )
     => t
```

Description

Validates the constraints supported by constraint-aware editing mode in the specified cellview.

Arguments

d_cellviewID	Database ID of the cellview for which constraints are to be validated.

Value Returned

`t`	Constraint validation was successful.
`nil`	Constraint validation was unsuccessful.

Example

```
cv = dbFindOpenCellView()
nclRunConstraintValidation(cv)
```

Validates the constraints in the cellview returned by the `dbFindOpenCellView` function.

nclToggleCAEMode

```
nclToggleCAEMode(
     )
     => t
```

Description

Toggles constraint-aware editing mode, which causes figures in the layout to move in a constraint-correct fashion. For example, moving a device that is part of a symmetry constraint moves its constrained partner devices appropriately.

You can use the `constraintAwareEditing` environment variable to check whether constraint-aware editing is currently on or off.

Arguments

None

Value Returned

`t`	Constraint-aware editing mode was toggled.
`nil`	Constraint-aware editing mode was not toggled.

Related Topics

constraintAwareEditiing

soiEnableDrcDfm

```
soiEnableDrcDfm(
     )
     => t
```

Description

Enables the *DRC/DFM* tab in the Annotation Browser and the DRC/DFM commands in the Optimize menu in Layout XL.

Arguments

None

Value Returned

`t`	The DRC/DFM capabilities are enabled.

Example

```
soiEnableDrcDfm()
```

techGetLxExtractLayers

```
techGetLxExtractLayers(
     d_techFileID
     )
     => l_extractLayers / nil
```

Description

Returns the list of extract layers from the `virtuosoDefaultExtractorSetup` constraint group in the specified technology file. These layers are listed as `validLayers` in the `interconnect` subsection of the constraint group.

Arguments

d_techFileID	Database ID of the technology file.

Value Returned

l_extractLayers	List of the valid layers listed in the `virtuosoDefaultExtractorSetup` constraint group. The list has the following syntax: (*lt_layer …*) *lt_layer* is a layer name.
`nil`	The technology file does not exist, the constraint group is not in the technology file, or the constraint group does not list any valid layers.

Example

```
techGetLxExtractLayers( techFileID )
=> ( pWell nDiff pDiff poly1 cont metal1 via1 metal2 via2 metal3 )
```

Returns the list of layers from the `interconnect(validLayers)` subsection in the `virtuosoDefaultExtractorSetup` constraint group in the technology file identified by *techFileID*.

techGetLxNoOverlapLayers

```
techGetLxNoOverlapLayers(
     d_techFileID
     )
     => l_noOverlapLayers / nil
```

Description

Lists the pairs of layers that cannot overlap.

Arguments

d_techFileID	Database ID of the technology file.

Value Returned

l_noOverlapLayers	List of the layers that cannot overlap. The list has the following syntax. ((*lt_layer* ...) ...) *lt_layer* is the layer data.
nil	The technology file or the list does not exist.

Example

```
techGetLxNoOverlapLayers( techFileID )
=> ( ( "poly1" "ndiff" )
     ( "poly1" "diff" )
     ( "metal1" "pdiff" )
    )
```

Returns the pairs of layers that cannot overlap from the technology file identified by *techFileID*.

techIsLxExtractLayer

```
techIsLxExtractLayer(
     d_techFileID
     tx_layer
     )
     => t / nil
```

Description

Indicates whether the specified layer is listed as an extract layer in the `virtuosoDefaultExtractorSetup` constraint group in the specified technology file. The extract layers are listed under `validLayers` in the `interconnect` subsection of the constraint group.

Arguments

d_techFileID	Database ID of the technology file.
tx_layer	The layer to be checked. Valid Values: the layer name or number

Value Returned

`t`	The specified layer is in the list.
`nil`	The technology file does not exist, the `virtuosoDefaultExtractorSetup` constraint group is not in the technology file, or the layer is not listed as a valid layer in the constraint group.

Example

```
techIsLxExtractLayer( techFileID "metal1" )
=> t
```

Confirms that the `metal1` layer is listed in the `interconnect(validLayers)` subsection in the `virtuosoDefaultExtractorSetup` constraint group in the technology file identified by *techFileID*.

techIsLxNoOverlapLayer

```
techIsLxNoOverlapLayer(
     d_techFileID
     tx_layer1
     tx_layer2
     )
     => t / nil
```

Description

Indicates whether the specified pair of layers is a derived no overlap layer in the specified technology file. The order in which the layers is specified does not matter.

Arguments

d_techFileID	Database ID of the technology file.
tx_layer1	First of two layers to be checked. Valid Values: Layer name or number.
tx_layer2	Second of two layers to be checked. Valid Values: Layer name or number

Value Returned

t	The pair of layers comprises a derived no overlap layer.
nil	The technology file does not exist or the layer pair does not comprise a no overlap derived layer.

Example

```
techIsLxNoOverlapLayer( techFileID '( "diff" "poly1" ) )
=> t
```

Confirms that the layers `diff` and `poly1` comprise a no overlap derived layer in the technology file identified by *techFileID*.

techSetLxExtractLayer

```
techSetLxExtractLayer(
     d_techFileID
     tx_extractLayer
     )
     => t / nil
```

Description

Adds the specified layer to the list of extract layers in the `virtuosoDefaultExtractorSetup` constraint group in the specified technology file. The extract layers are listed under `validLayers` in the `interconnect` subsection of the constraint group. If any of the necessary technology file sections do not already exist, this function creates them.

Arguments

d_techFileID	Database ID of the technology file.
l_extractLayers	Layer name or number.

Important

To optimize performance, specify only the name of each layer to extract, not the layer name and layer purpose. If you specify a purpose, it is ignored.

Value Returned

`t`	The specified layer was added.
`nil`	The technology file does not exist, the constraint group is not in the technology file, or the constraint group does not list any valid layers.

Example

```
techSetLxExtractLayer( techFileID "metal1" )
```

Adds the `metal1` layer to the list of extract layers in the `virtuosoDefaultExtractorSetup` constraint group in the technology file identified by *techFileID*.

techSetLxExtractLayers

```
techSetLxExtractLayers(
     d_techFileID
     l_extractLayers
     )
     => t / nil
```

Description

Adds the specified layers to the list of extract layers in the `virtuosoDefaultExtractorSetup` constraint group in the specified technology file. The extract layers are listed under `validLayers` in the `interconnect` subsection of the constraint group. If any of the necessary technology file sections do not already exist, this function creates them.

Arguments

d_techFileID	Database ID of the technology file.
l_extractLayers	List of extract layers. The list has the following syntax: `list ( tx_layer … )` *tx_layer* is the layer name or number.

Important

To optimize performance, specify only the name of each layer to extract, not the layer name and layer purpose. If you specify a purpose, it is ignored.

Value Returned

`t`	The list of extract layers was added.
`nil`	The technology file does not exist, the constraint group is not in the technology file, or the constraint group does not list any valid layers.

Example

```
techSetLxExtractLayers( techFileID list( "metal1" "poly" "metal2" ) )
```

Adds the specified layers to the `virtuosoDefaultExtractorSetup` constraint group in the technology file identified by *techFileID*.

techSetLxNoOverlapLayer

```
techSetLxNoOverlapLayer(
     d_techFileID
     tx_layer1
     tx_layer2
     )
     => t / nil
```

Description

Derives a no overlap layer by performing an AND operation on the two specified layers, and flags it as an error layer in the `virtuosoDefaultExtractorSetup` constraint group in the specified technology file. The derived layer is defined in the `techDerivedLayers` subsection in the `layerDefinitions` section of the technology file and is listed as an `errorLayer` in the `interconnect` subsection of the constraint group. No overlap layers are named and numbered consecutively; each is named `noOverlapLayer`*n*, where *n* is the next available consecutive integer. If any of the necessary technology file sections does not already exist, the function creates them.

Arguments

d_techFileID	Database ID of the technology file.
tx_layer1	First of two layers that cannot overlap. Valid Values: Layer name or number.
tx_layer2	Second of two layers that cannot overlap. Valid Values: Layer name or number

Value Returned

`t`	The derived layer was created, defined in the `techDerivedLayers` subsection, and listed as an error layer in the constraint group.
`nil`	The technology file does not exist.

Example

```
techSetLxNoOverlapLayer( techFileID ( "poly1" "ndiff" ) )
```

Creates the derived layer and lists it as an error layer in the constraint group `virtuosoDefaultExtractorSetup` in the technology file identified by *techFileID*.

techSetLxNoOverlapLayers

```
techSetLxNoOverlapLayers(
     d_techFileID
     l_noOverlapLayers
     )
     => t / nil
```

Description

Creates a list of layers that cannot overlap. The function creates a derived layer by performing an AND operation on each pair of layers and flags the resultant layer as an error layer in the `virtuosoDefaultExtractorSetup` constraint group in the specified technology file. Each derived layer is defined in the `techDerivedLayers` subsection in the `layerDefinitions` section of the technology file and is listed as an `errorLayer` in the `interconnect` subsection of the constraint group.

No overlap derived layers are named and numbered consecutively; each is named `noOverlapLayer`*`n`*, where *`n`* is the next available consecutive integer. If any of the necessary technology file sections does not already exist, this function creates them.

Arguments

d_techFileID

Database ID of the technology file.

l_noOverlapLayers

A list of lists of layer pairs that cannot overlap. The list has the following syntax.

`list ( list ( tx_layer1 tx_layer2 ) … )`

tx_layer1 is first of two layers that cannot overlap.
Valid Values: Layer name or number

tx_layer2 is second of two layers that cannot overlap.
Valid Values: Layer name or number

Value Returned

`t`	The derived layers were created, defined in the `techDerivedLayers` subsection, and listed as error layers in the constraint group.
`nil`	The technology file does not exist.

Examples

```
techSetLxNoOverlapLayers( techFileID list( list( "poly1" "ndiff" )
list( "poly1" "diff" ) list( "metal1" "pdiff" ) )
=> t
```

Creates three derived layers in the technology file identified by *techFileID*.

```
techDerivedLayers(
; ( DerivedLayerName     #        composition))
  ( noOverlapLayer1      10001    ( poly1   'and   pdiff ))
  ( noOverlapLayer2      10002    ( poly1   'and   diff  ))
  ( noOverlapLayer3      10003    ( metal1  'and   pdiff ))
;techDerivedLayers
```

Shows the `techDerivedLayers` subsection in the `layerDefinitions` section. Assuming that these are the first three no overlap derived layers created, they would be named `noOverlapLayer1`, `noOverlapLayer2`, and `noOverlapLayer3` and are listed as error layers in the `virtuosoDefaultExtractorSetup` constraint group.

```
("virtuosoDefaultExtractorSetup"   nil

  interconnect(
```

```
   ( validLayers   (cont  nwell  metal3  metal2  via2  via  metal1  poly1  diff
    pdiff  ndiff  isolation  buried  pwell ) )
   ( errorLayer    noOverlapLayer1 )
   ( errorLayer    noOverlapLayer2 )
   ( errorLayer    noOverlapLayer3 )
) ;interconnect

);virtuosoDefaultExtractorSetup
```

tpoIsLayoutXL

```
tpoIsLayoutXL(
     d_windowID
     )
     => t / nil
```

Description

Checks if the specified window is open in Layout XL or a higher tier.

Arguments

d_windowID

Database ID of the window.

Value Returned

`t`	The specified window is open in Layout XL or a higher tier.
`nil`	The specified window is not open in Layout XL or a higher tier.

Examples

```
when(window = hiGetCurrentWindow()
  tpoIsLayoutXL(window)
)
```

tpoIsLayoutEXL

```
tpoIsLayoutEXL(
     d_windowID
     )
     => t / nil
```

Description

Checks if the specified window is open in Layout EXL or a higher tier.

Arguments

d_windowID

Database ID of the window.

Value Returned

`t`	The specified window is open in Layout EXL or a higher tier.
`nil`	The specified window is not open in Layout EXL or a higher tier.

Examples

```
when(window = hiGetCurrentWindow()
  tpoIsLayoutEXL(window)
)
```

3

Fluid Guard Ring Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso Fluid Guard Rings (FGR).

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- vfoAbut
- vfoAdvGRUpdateCDF
- vfoAllocateProtocolObj
- vfoAllocateShapeData
- vfoChopInstance
- vfoConvertToPolygon
- vfoCreateAutoFGR
- vfoCreateObstruction
- vfoCreateObstructions
- vfoCreateShapeData
- vfoDeleteObstruction
- vfoDoSnapToSnapPattern
- vfoDrawFluidShape
- vfoGetDeviceClassParam
- vfoGetDeviceFormalParam
- vfoGetFileListWithLoadSequence
- vfoGetImplementationClassName

- vfoGetInstWithMissingCache
- vfoGetParam
- vfoGetProtocolClassName
- vfoGetVersion
- vfoGRCleanVersionCache
- vfoGrCreateCDF
- vfoGRCompareParams
- vfoGRCreateDeviceClass
- vfoGRDisableVersionCache
- vfoGREnableVersionCache
- vfoGRGetCreateFormFieldProp
- vfoGRGetCreateFormIdentifier
- vfoGRGetCreateFormPointer
- vfoGRGetCommonQPtr
- vfoGRGetExtraArgument
- vfoGRGetExtraArgumentName
- vfoGRGetQueuePointer
- vfoGRGeometry
- vfoGRMaximizeShapes
- vfoGRNewCreateForm
- vfoGRRegCreateFormUpdateCallback
- vfoGRSetCreateFormAllFieldsInvisible
- vfoGRSetCreateFormFieldProp
- vfoGRSetExtraArgument
- vfoGRSmoothen
- vfoGRUpdateCreateFormSize
- vfoGRUpdateTunnelLPPs

- vfoGRUpdateTunnelOptions
- vfoGRUpdateVersionCache
- vfoInstallFluidDeviceFiles
- vfoIsCommandInDragMode
- vfoIsGuardRing
- vfoMergeInstances
- vfoPostChopCBHandler
- vfoPostReshapeCBHandler
- vfoPostSplitCBHandler
- vfoPostStretchCBHandler
- vfoReInstallAllFluidGuardRingDevices
- vfoReInstallGuardRingDevice
- vfoRotateInstance
- vfoSetParam
- vfoSetParams
- vfoSetProtocolClassName
- vfoSfDraw
- vfoSfFinalize
- vfoSfInitialize
- vfoSfOuterEdgeCornerContRemoval
- vfoSmoothen
- vfoSupportsAbut?
- vfoSupportsChop?
- vfoSupportsConvertToPolygon?
- vfoSupportsCreateObstruction?
- vfoSupportsCreateObstructions?
- vfoSupportsDeleteObstruction?

- vfoSupportsMerge?
- vfoSupportsRotation
- vfoSupportsUpdateModelShape?
- vfoSupportsVersionCache
- vfoTransformFluidShape
- vfoTunnelHigherMetalLayers
- vfoUpdateModelShape
- xfgrAddOption
- xfgrAddRow
- xfgrAddTable
- xfgrConfigureOptionsCB
- xfgrDeleteRow
- xfgrDisableOption
- xfgrEnableOption
- xfgrGetAvailableValues
- xfgrGetCellProps
- xfgrGetColumnProps
- xfgrGetFormSize
- xfgrGetGroupProps
- xfgrGetOptionProps
- xfgrGetOptionValue
- xfgrGetTableProps
- xfgrGetTotalTableRows
- xfgrHideAllGroups
- xfgrHideAllOptions
- xfgrHideGroup
- xfgrHideOption

- xfgrInitializeOptionsCB
- xfgrOptionValueChangeCB
- xfgrPostCreateCB
- xfgrPreCreateCB
- xfgrPrintTable
- xfgrResetTable
- xfgrSetAvailableValues
- xfgrSetCellProps
- xfgrSetColumnProps
- xfgrSetFormSize
- xfgrSetGroupProps
- xfgrSetOptionProps
- xfgrSetOptionValue
- xfgrSetTableCellValue
- xfgrTableCellValueChangedCB
- xfgrSetTableProps
- xfgrTableRowAddedCB
- xfgrTableRowDeletedCB
- xfgrShowAllGroups
- xfgrShowGroup
- xfgrShowOption

Related Topics

Virtuoso Fluid Guard Ring User Guide

vfoAbut

```
vfoAbut(
     g_obj
     l_instsList
     [ @rest l_args ]
     )
     => t / nil
```

Description

Abuts the instances provided in the *l_instList*. You can pass optional arguments to this SKILL API.
Note: The *Abut* command does not call the `vfoAbut` function for fluid devices.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
l_instsList	List of database IDs of fluid Pcell instances on which the abutment operation is called.
l_args	List of optional arguments that can be passed to the function.

Value Returned

`t`	The instances were successfully abutted.
`nil`	The instances could not be abutted.

Example

```
Obj = makeInstance('vfoAbstractClass)
    => stdobj:0x325030
vfoAbut(Obj list(inst1 inst2))
    => t
```

vfoAdvGRUpdateCDF

```
vfoAdvGRUpdateCDF(
     g_obj
     t_libName
     t_cellName
     t_viewName
     )
     => l_cdfParams
```

Description

Helps to modify the existing CDFs on a FGR device and add more CDFs on it. The function can be defined for subclasses of `vfoAdvGuardRing`. It works based on callback mechanism, that is, the user defines it and the infrastructure calls it.

Arguments

g_obj	An instance of the `vfoAdvGuardRing` SKILL++ class or any of its subclasses.
t_libName	The name of the technology library containing the FGR device.
t_cellName	The name of the cell in which the FGR device exists.
t_viewName	The name of the view associated to the FGR device.

Value Returned

l_cdfParams	List of user-defined CDF parameters.

Example

```
defclass(vfoAdvGRExtended (vfoAdvGuardRing) () )
defmethod(vfoAdvGRUpdateCDF ( (obj vfoAdvGRExtended ) libName cellName viewName)
let((cellId cdfId paramId cvId)
    cellId = ddGetObj(libName cellName)
    cdfId = cdfGetBaseCellCDF(cellId)
    unless(cdfId
        cdfId = cdfCreateBaseCellCDF(cellId)
    );;unless
    cvId = dbFindOpenCellView(ddGetObj(libName) cellName viewName)
    ;create user defined parameter
```

```
    when(cdfFindParamByName(cdfId "userCDF")
        cdfDeleteParam(cdfFindParamByName(cdfId "userCDF"))
    )

    cdfCreateParam(cdfId
        ?name           "userCDF"
        ?prompt         "User CDF"
        ?defValue       0.0
        ?value          0.0
        ?type           "float"
        ?display        "t"
    )
    putprop(cdfId->readers '(lambda (x) x) "topMetal")
    cdfSaveCDF(cdfId)
)
```

vfoAllocateProtocolObj

```
vfoAllocateProtocolObj(
     d_obj
     )
     => g_instance / nil
```

Description

Allocates and returns the newly allocated and initialized instance of the vfo protocol class of the Pcell. You can use the vfoGetProtocolClassName SKILL function to determine the name of the protocol class.

Arguments

d_obj	The database ID of an instance of a fluid Pcell or the Pcell master.

Value Returned

g_instance	Instance of the protocol class whose name is stored on the d_obj Pcell supermaster.
nil	Class object is not returned if the protocol class property is not set on the d_obj Pcell supermaster.

Example

```
Obj = vfoAllocateProtocolObj(d_inst )
    => stdobj:0x325030
```

vfoAllocateShapeData

```
vfoAllocateShapeData(
     d_shape
     )
     => g_instance / nil
```

Description

Returns a newly allocated and initialized instance of the subclass of the `vfoShapeData` class for the specified database shape ID.

Arguments

d_shape	The database shape ID of one of the following: path, rectangle, polygon, or line.

Value Returned

g_instance	An instance of a subclass of `vfoShapeData`.
`nil`	The shape data object is not returned.

Example

```
Shape1=>objType = "rect"
Obj = vfoAllocateShapeData (Shape1)
    => stdobj:0x1d003c
className( classOf( Obj ))
    => vfoRectShapeData

Shape2=>objType = "polygon"
Obj = vfoAllocateShapeData (Shape2)
    => stdobj:0x1d003c
className( classOf( Obj ))
    => vfoPolygonShapeData
```

vfoChopInstance

```
vfoChopInstance(
     g_obj
     d_instId
     chopShapeData
     [ @rest l_args ]
     )
     => t / nil
```

Description

Chops the fluid shape of a given instance. The information of the chop rectangle or polygon is provided by `chopShapeData` which is a class object of `vfoShapeData`. The shape data points are in the design coordinates. This function transforms the `chopShapeData` before chopping the fluid shape.

Note: The `Chop` command for fluid Pcells does not pass these arguments. This function updates the relevant Pcell parameters. If the chop operation results in multiple shapes, it may need to create multiple Pcell instances.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.
chopShapeData	Instance of `vfoShapeData` (or any of its subclasses) class.
l_args	List of optional arguments that can be passed to the function.

Value Returned

`t`	The fluid shape of the given instance has been successfully chopped.
`nil`	The fluid shape of the given instance could not be chopped.

Example

Create a shape data object as following:

```
shapeData = vfoCreateShapeData("rect" '((2 2) (3 1)))
    => stdobj@0x148e90d8
obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148e90e4
vfoChopInstance(obj inst shapeData)
    => t
```

In the above example, the points of the chop shape are specified using the `shapeData` function. `inst` is the fluid Pcell instance to be chopped.

vfoConvertToPolygon

```
vfoConvertToPolygon(
     g_obj
     d_instId
     [ @rest l_args ]
     )
     => t / nil
```

Description

Converts the fluid shape of the given instance to a polygon. This is used for instances containing path type fluid shape. The *Convert to Polygon* command for fluid Pcells does not pass these arguments. This function updates the relevant Pcell parameters.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.
l_args	List of optional arguments that can be passed to the function.

Value Returned

`t`	The fluid shape has been converted to a polygon.
`nil`	The fluid shape could not be converted to a polygon.

Example

In the following example, `inst` is the fluid Pcell instance to be converted.

```
obj = vfoAllocateProtocolObj(inst)
=> stdobj@0x148e90e4
vfoConvertToPolygon(obj inst)
    => t
```

vfoCreateAutoFGR

```
vfoCreateAutoFGR(
     )
     => nil
```

Description

Wraps obstacles in wrap mode on clicking an obstacle if *AUTO* mode is selected in the *Device* field of the Create Guard Ring form. This function is used to specify the bindkey for creation of FGR in auto mode.

Arguments

None

Value Returned

`nil`	This is the value returned for this function.

Example

```
hiSetBindKey("Layout" "Ctrl Shift<Key>f" "vfoCreateAutoFGR")
```

If you press the key combination, `Ctrl+Shift+f`, FGR is created in AUTO mode.

vfoCreateObstruction

```
vfoCreateObstructions(
     g_obj
     d_instId
     layerPurposePair
     obstructionShapeData
     [ @rest l_args ]
     )
     => t / nil
```

Description

Creates obstruction or tunnel on a given instance that has an associated fluid shape. A tunnel helps you to cut a specific layer-purpose of a guard ring device for connectivity purpose. There are four types of tunneling mechanisms—Path, Rectangle, Polygon, and Overlapping Shape. For procedural details about creating tunnels using GUI, refer to the *Creating a Tunnel Through a Fluid Guard Ring* section in the *Editing Fluid Guard Rings* chapter of the *Virtuoso Fluid Guard Ring User Guide*.

Note: To remove any existing obstructions, use the vfoDeleteObstruction SKILL function.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.
layerPurposePair	The layer-purpose pair on which the obstruction needs to be created.
obstructionShapeData	The information of the fluid shape associated to the instance on which the obstruction needs to be created. It can be specified either as a SKILL++ object of the `vfoShapeData` class or a simple point list in case of FGRs. The shape data points are in the design coordinates. The specified *obstructionShapeData* argument is transformed before creating the obstruction.

l_args	List of optional arguments that can be passed to the function. While creating an obstruction on an FGR instance that has overlapping fluid shapes, use the *l_args* argument to define the spacing parameters to increase the size of the shape data points specified with the `obstructionShapeData` argument. In this case, the SKILL function definition is equivalent to selecting the *Overlapping Shape* radio button on the *Create Tunnel in Fluid Object* form.

Value Returned

`t`	The obstruction was created successfully.
`nil`	The obstruction could not be created.

Example

- Creates obstruction on the fluid Pcell instance, `inst`:

```
obj = vfoAllocateProtocolObj(inst)
    stdobj@0x148e90e4
vfoCreateObstruction(obj inst `("OD" "drawing") list((3 3) (3 6) (5 6) (5 3)) )
    t
```

- Creates obstruction on the FGR instance, `fgrInst`, that has overlapping fluid shapes:

```
vfoCreateObstruction(obj fgrInst ("Metal1" "drawing") ((219.5 79.8) (221.7
79.8) (221.7 80.8) (219.5 80.8)) 0.5)
    t
```

vfoCreateObstructions

```
vfoCreateObstructions(
     g_obj
     d_instId
     l_lppPoint
     [ @rest l_args ]
     )
     => nil
```

Description

Creates obstruction or tunnel on a given instance that has an associated fluid shape. A tunnel helps you to cut a specific layer-purpose of a guard ring device for connectivity purpose. There are four types of tunneling mechanisms—Path, Rectangle, Polygon, and Overlapping Shape. For procedural details about creating tunnels using GUI, refer to the *Creating a Tunnel Through a Fluid Guard Ring* section in the *Editing Fluid Guard Rings* chapter of the *Virtuoso Fluid Guard Ring User Guide*.

Note: To remove any existing obstructions, use the vfoDeleteObstruction SKILL function.

Arguments

g_obj	An instance of the `vfoSfImplEditClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.
l_lppPoint	List of pairs in the form (LPP pointlist) for obstruction.
l_args	List of optional arguments that can be passed to the function. While creating an obstruction on an FGR instance that has overlapping fluid shapes, use the *l_args* argument to define the spacing parameters to increase the size of the shape data points specified with the `obstructionShapeData` argument. In this case, the SKILL function definition is equivalent to selecting the *Overlapping Shape* radio button on the *Create Tunnel in Fluid Object* form.

Value Returned

`nil`	This is the value returned for this function.

vfoCreateShapeData

```
vfoCreateShapeData(
     t_shapeType
     [ @rest l_args ]
     )
     => g_instance / nil
```

Description

Allocates, initializes, and returns an instance of the subclass of the `vfoShapeData` of the given shape type.

Arguments

t_shapeType	The string that describes the shape type. This can be `path`, `rectangle`, `polygon`, or `line`.
l_args	For `path` shape type, provide the list (width pointlist). For the `rectangle` shape type, provide the bbox values. For the `polygon` and `line` shape types, provide the pointlist.

Value Returned

g_instance	An initialized SKILL++ object of a subclass of the `vfoShapeData` is returned.
`nil`	The SKILL++ object of a subclass of the `vfoShapeData` is not returned.

Example

In the above example, the points of the chop shape are specified using the `shapeData` function.

```
shapeData = vfoCreateShapeData("rect" '((2 2) (3 1)))
    => stdobj@0x148e90d8
className(classOf(shapeData))
    => vfoRectShapeData
shapeData = vfoCreateShapeData("polygon" '((5 0) (0 0) (0 6)))
className(classOf(shapeData))
    => vfoPolygonShapeData
```

vfoDeleteObstruction

```
vfoDeleteObstruction(
     g_obj
     d_instId
     layerPurposePair
     obstructionShapeData
     [ @rest l_args ]
     )
     => t / nil
```

Description

Deletes the obstruction or tunnel on a given instance that has an associated fluid shape. This SKILL function is equivalent to healing.

For procedural details about healing an obstruction using GUI, refer to the *Healing a Fluid Guard Ring* section in the *Editing Fluid Guard Rings* chapter of the *Virtuoso Fluid Guard Ring User Guide*.

Note: To create an obstruction, use the vfoCreateObstruction SKILL function.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.
layerPurposePair	The list of layer-purpose pairs from which the obstruction needs to be deleted.
obstructionShapeData	

The information of the fluid shape associated to the instance from which the obstruction needs to be deleted. The following possible values can be specified:

- `nil`: Deletes all the obstructions. This does not consider the specified layer purpose pair and deletes all the obstructions.
- `point`: Deletes the obstructions at that specific point. With this value, you need to specify the list of layer purpose pairs from which the obstruction needs to be removed.
- `bbox`: Deletes the obstruction overlapping the given bbox. With this value, you need to specify the list of layer purpose pairs from which the obstruction needs to be removed.
- `polygon`: Deletes the obstruction overlapping the given polygon. With this value, you need to specify the list of layer purpose pairs from which the obstruction needs to be removed.

`l_args` — List of optional arguments that can be passed to the function.

Note: Currently, the `@rest` *`l_args`* argument cannot be used while deleting an obstruction from an FGR instance. It has been provided for the flexibility of performing any user-defined processing if you had previously used the vfoCreateObstruction SKILL function.

Value Returned

`t`	The obstruction was deleted successfully.
`nil`	The obstruction could not be deleted.

Example

Assume, `inst` is the fluid Pcell instance that needs to be healed by removing the obstructions:

```
obj = vfoAllocateProtocolObj(inst)
```

Then, the following examples show the different ways of using the `vfoDeleteObstruction` SKILL function:

- Removes all the keepouts of the instance:

```
"vfoDeleteObstruction(obj inst list("Metal1" "drawing") nil)
```

- Removes all the keepouts at point, `pt`, of the layer purpose pair:

```
"vfoDeleteObstruction(obj inst list(list("Oxide" "drawing") list("Metal1"
"drawing")) list(pt))
```

- Removes all the keepouts at bbox, `b`, of the layer purpose pair. The bbox should be completely inside the keepout region.

```
"vfoDeleteObstruction(obj inst list(list("Oxide" "drawing") list("Metal1"
"drawing")) b)
```

- Removes all the keepouts at point list, `ptList`, of any polygon (more than two points) of the layer purpose pair. The polygon points should be completely inside the keepout region.

```
"vfoDeleteObstruction(obj inst list(list("Oxide" "drawing") list("Metal1"
"drawing")) ptList)
```

vfoDoSnapToSnapPattern

```
vfoDoSnapToSnapPattern (
     d_inst
     )
     => t / nil
```

Description

Snaps the fluid object to the snap pattern.

Arguments

d_inst	The ID of the fluid object instance.

Value Returned

t	The fluid object was snapped to the snap pattern.
nil	The fluid object could not be snapped to the snap pattern.

Example

```
vfoDoSnapToSnapPattern(objFluid)
```

vfoDrawFluidShape

```
vfoDrawFluidShape(
     g_shapeObj
     cellView
     layerPurposePair
     )
     => d_obj / nil
```

Description

Creates a `dbObject` corresponding to `shapeData` object in the given cellview on the given layer purpose pair.

Argumentsd

shapeObj	SKILL++ object of any subclass of `vfoShapeData`.
cellView	Cellview in which the database object needs to be created.
layerPurposePair	The layer purpose pair to be used for the database object.

Value Returned

d_obj	The database ID of the new object is returned.
`nil`	The database ID of the new object could not be returned.

Example

```
shapeData = vfoCreateShapeData("rect" list(0:0 10:10))
    => stdobj@0x148f51e0
vfoDrawFluidShape(shapeData cvId '("OD" "drawing"))
    => db:0xf11ec912
```

vfoGetDeviceClassParam

```
vfoGetDeviceClassParam(
     t_libName
     t_deviceName
     t_classParam
     )
     => value / nil
```

Description

Returns the class parameter of the specified `cdsGuardRing` device. This SKILL function also supports the incremental technology database (ITDB). Therefore, the information is retrieved for FGR devices in both the library and the referenced library.

Arguments

t_libName	The name of the technology library containing the FGR device.
t_deviceName	The name of the FGR device.
t_classParam	The name of the class parameter.

Value Returned

value	The value of the class parameter in device definition.
`nil`	The value of the class parameter could not be found in the device definition.

Example

```
vfoGetDeviceClassParam(libName devName "vfoGRImpl")
```

vfoGetDeviceFormalParam

```
vfoGetDeviceFormalParam(
     t_libName
     t_deviceName
     t_formalParam
     )
     => value / nil
```

Description

Returns the implementation formal parameter of the specified `cdsGuardRing` device. This SKILL function also supports ITDB. Therefore, the information is retrieved for FGR devices in both the library and the referenced library.

Arguments

t_libName	The name of the technology library containing the FGR device.
t_deviceName	The name of the FGR device.
t_formalParam	The name of the formal parameter.

Value Returned

value	The value of the formal parameter in device definition.
`nil`	The value of the formal parameter could not be found in the device definition.

Example

```
vfoGetDeviceFormalParam(libName devName "xContSpacing")
```

vfoGetFileListWithLoadSequence

```
vfoGetFileListWithLoadSequence(
     )
     => l_vfoFilename
```

Description

Returns a list of `vfo*.ils` filenames with their loading sequence.

Arguments

None

Value Returned

`l_vfoFilename` A list of `vfo*.ils` filenames.

Example

```
vfoGetFileListWithLoadSequence()
("vfoMessageIds.ils" "vfoAbstractClass.ils" "vfoAddOns.ils" "vfoApi.ils"
    "vfoAlgClass.ils" vfoUtils.ils" "vfoShapeData.ils" "vfoSfShapeData.ils"
    "vfoSf.ils" "vfoSfFilling.ils" "vfoGuardRing.ils" "vfoGrShrinkWrap.ils"
    "vfoGuardRingPreview.ils" "vfoTriggers.ils"
)
```

vfoGetImplementationClassName

```
vfoGetImplementationClassName(
     t_libName
     t_deviceName
     )
     => value
```

Description

Returns the value of the `vfoGRImpl` class parameter, that is the implementation class name, of an FGR device from the specified library. This SKILL function also supports the incremental technology database (ITDB). Therefore, the information is retrieved for FGR devices in both the library and the referenced library.

Arguments

libName	The name of the technology library containing the FGR device.
deviceName	The name of the FGR device.

Value Returned

value	The value of the `vfoGRImpl` class parameter is returned.

Example

Suppose, you have a device, `Nring`, created in library, `reflib`. The associated implementation class, `myImpClass`, can be returned by running the following SKILL function in the CIW:

```
vfoGetImplementationClassName("reflib" "Nring")
```

vfoGetInstWithMissingCache

```
vfoGetInstWithMissingCache(
     t_libName
     t_cellName
     t_viewName
     )
     => l_dbIDs
```

Description

Returns three lists of dbIDs. The first is the list of dbIDs of all instances whose submaster is not found in index.pcl file. The second is the list of dbIDs of all instances whose submaster is found in index.pcl file but not found in cache.pcl file because the file is corrupt or missing. The third is the list of dbIDs of instances whose submaster is not found because the supermaster has been updated after instance creation.

Note: The cellview should be closed while using this function.

Arguments

t_libName	The name of the design library in which you want to search instances with missing cache.
t_cellName	The name of the cellview in which you want to search instances with missing cache.
t_viewName	The name of the view in which you want to search instances with missing cache.

Value Returned

l_dbIds	Returns the list of three sub-lists of dbIDs. The first sub-list contains dbIDs of instances whose submaster is not found in index.pcl file, the second sub-list contains dbIDs of instances whose cache is found corrupt, and the third sub-list contains dbIDs of instances whose supermasters were updated after the instance creation so no cache was found. For example, ((db:0x2a04aa2a) (db:0x2a04dd9a db:0x2a04dd9b) (db:0x2a04dd8c))

Example

Get the list of instances in design `myLib/myCell/layout` whose FGR cache is not found in the cache files.

```
design = dbOpenCellViewByType( "myLib" "myCell"  "layout" "a" )
instanceList = vfoGetInstWithMissingCache( "myLib"  "myCell"  "layout" )
```

; iterate over the first list of instances and set latest FGR cache version to update the FGR cache.

```
(foreach inst car(instancelist)
       vfoSetParam(inst "cacheCreateVersion" vfoGetVersion())
)
(foreach inst car(cdr(instancelist))
       vfoSetParam(inst "cacheCreateVersion" vfoGetVersion())
)
(foreach inst car(cdr(cd(instancelist)))
       vfoSetParam(inst "cacheCreateVersion" vfoGetVersion())
)
```

; save the design to update FGR cache.

```
dbSave(design)
dbPurge(design)
```

vfoGetParam

```
vfoGetParam(
     g_obj
     s_param
     )
     => value / nil
```

Description

Returns the value of a CDF parameter for the given object.

Arguments

g_obj	The object for which the CDF parameter value should be returned. The object can be one of the following: an instance database object, a cell CDF, or an instance CDF.
s_param	The CDF parameter for which the value needs to be returned.

Value Returned

value	The CDF parameter value is returned.
nil	The CDF parameter value could not be returned.

Example

```
vfoGetParam(inst "shapeData")
    =>((5.39 1.19)
        (0.78 1.19)
        (0.78 6.09)
        (0.0 6.09)
        (0.0 0.0)
        (5.39.0)
        )
vfoGetParam(inst "shapeType")
    => "polygon"
```

vfoGetProtocolClassName

```
vfoGetProtocolClassName(
     t_cellViewID
     )
     => s_cellView / nil
```

Description

Returns the symbol naming the protocol class to use with the fluid Pcell functions.

Argumentsd

t_cellViewID	The ID of the Pcell sub-master or supermaster cellview.

Value Returned

s_cellView	The symbol naming the protocol class to use with the fluid Pcell functions is returned.
nil	The protocol class property not found in the cellview.

Example

```
vfoGetProtocolClassName(inst ~> master)
    => vfoAbstractClass
vfoGetProtocolClassName(inst ~> master ~> superMaster)
    => vfoAbstractClass
```

vfoGetVersion

```
vfoGetVersion(
     )
     => t_version
```

Description

Returns the version of the VFO SKILL code used in Virtuoso. This SKILL function is available from IC6.1.6 ISR2 and ICADV12.1 ISR4 onwards.

Arguments

None

Value Returned

t_version Returns the version information.

Example

If you are working on IC6.1.6 ISR2 version of Virtuoso, the following SKILL function given in the CIW will return `FGR_616.2`:

```
vfoGetVersion()
=> FGR_616.2
```

vfoGRCleanVersionCache

```
vfoGRCleanVersionCache(
     t_libName
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     )
     => t / nil
```

Description

Updates the `index.pcl` and `cache.pcl` files by removing any unused entries of sub-masters from these files. This is an alternative to cleaning the cache if you have disabled the automatic FGR cache cleaning mechanism described in the Automatic Cache Cleaning section, or even if the Automatic Cache Cleaning mechanism is enabled by you and would like to clean the FGR caches on-demand.

Important

You should ensure that you save any changes done in the layout cellview before you run this SKILL function.

Arguments

t_libName Specifies the library name for which you want to run FGR cache cleaning.

?cellName *t_cellName*

If a cell name is specified, only cells with the given name are processed and all others are ignored. All views for the given cell name are processed, and `cache.pcl` and `index.pcl` files are updated in all views of the specified cell name. If no cell name is specified, all cells are processed, and `cache.pcl` and `index.pcl` are updated.

?viewName *t_viewName*

If a view name is specified, only the specified view name is processed for all cells specified in the `t_cellName` argument. If a view name is not specified, all views of each cell are processed, and `index.pcl` and `cache.pcl` files are updated.

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Example

If the `index.pcl` file contain unused entries of sub-masters, the `vfoGRCleanVersionCache` function cleans the unused entries, and the file size will be reduced.

```
fileLength("reflib/test1/layout2/cache.pcl")
>111196
fileLength("reflib/test1/layout2/index.pcl")
>3624

vfoGRCleanVersionCache("reflib" ?cellName "test1")

fileLength("reflib/test1/layout2/cache.pcl")
>35408
fileLength("reflib/test1/layout2/index.pcl")
>1218
```

vfoGRCompareParams

```
vfoGRCompareParams(
     g_obj
     d_inst1
     d_inst2
     )
     => t / nil
```

Description

Checks the equality of FGR instances. You override this function and can define its new equality for FGR instances. By default, it checks the value of the following parameters: `verticalPitch`, `horizontalPitch`, `verticalSegWidth`, and `horizontalSegWidth`. This function is called by the *Merge* command. For the *Merge* command to succeed, this function should return `t` for the two FGR instances being compared.

Arguments

g_obj	An instance of the `vfoAdvGuardRing` SKILL++ class or any of its subclasses.
d_inst1 *d_inst2*	The database ID of the two FGR instances that need to be compared.

Value Returned

`t`	The values of the following four parameters are equal for the two specified FGR instances: `verticalPitch`, `horizontalPitch`, `verticalSegWidth`, and `horizontalSegWidth`.
`nil`	The values of the four parameters mentioned above are not equal for the two specified FGR instances.

Example

```
defmethod(vfoGRCompareParams
    ((obj vfoAdvGuardRing) inst1 inst2)
    let(()
    if( ((vfoGetParam(inst1 "verticalPitch") ==
        vfoGetParam(inst2 "verticalPitch")) &&
        (vfoGetParam(inst1 "horizontalPitch") ==
```

```
        vfoGetParam(inst2 "horizontalPitch")) &&
        (vfoGetParam(inst1 "horizontalSegWidth") ==
        vfoGetParam(inst2 "horizontalSegWidth")) &&
        (vfoGetParam(inst1 "verticalSegWidth") ==
        vfoGetParam(inst2 "verticalSegWidth"))) t nil)
    )
)
```

vfoGrCreateCDF

```
vfoGrCreateCDF(
     t_libName
     t_cellName
     t_viewName
     )
     => l_cdfParams
```

Description

Creates fluid guard ring CDF parameters on the given library, cell view of the fluid guard ring device. It deletes any existing CDF parameters on the device.

Arguments

t_libName	The name of the technology library containing the FGR device.
t_cellName	The name of the FGR device.
t_viewName	The name of the view for the device. Typically, the value will be `"layout"`.

Value Returned

l_cdfParams	List of CDF parameters created.

Example

```
vfoGrCreateCDF("PDKLIB" "N_GR" "layout")
    => (cdf:0x0x19919360 cdf:0x0x19919378 cdf:0x0x19919390)
```

vfoGRCreateDeviceClass

```
vfoGRCreateDeviceClass(
     t_techLibName
     [ ?override g_override ]
     )
     => t / nil
```

Description

Creates the device class definition for fluid guard ring (`cdsGuardRing`) in the given technology file. If the device class already exists in the technology file, it is updated when override is set to true.

Arguments

t_techLibName	The name of the technology file name.
?override *g_override*	A Boolean value that indicates if the device class definition should be updated in the technology file. The default value is `nil`.

Value Returned

`t`	The device class is successfully created.
`nil`	The device class could not be created.

Example

```
vfoGRCreateDeviceClass("PDKLIB" ?override t)
    => t
```

vfoGRDisableVersionCache

```
vfoGRDisableVersionCache(
     l_FGRcellDDID
     t_libName
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     )
     => t / nil
```

Description

Disables version cache for all instances that belong to the specified super-master list and found in the specified library, cell, or view. All instances are updated and the value of their `createVersionCache` parameter is set to `0`.

Note: The function `vfoGRDisableVersionCache` calls the `dbSave` function in the background, so any unsaved changes in the layout cellview are saved when the `vfoGRDisableVersionCache` function is called.

Arguments

l_FGRcellDDID	The list of dd IDs, returned by the ddGetObj() function, for the fluid guard ring cells to be updated.
t_libName	The library name in which you want to disable version cache for the FGR instances found in the applicable layout cellviews of the library.

?cellName *t_cellName*

If a cell name is specifed, only cells with the given name are processed and all others are ignored. All views for the given cell name are processed, and cache.pcl and index.pcl files are updated in all views of the specified cell name. If no cell name is specified, all cells are processed, and cache.pcl and index.pcl are updated.

?viewName *t_viewName*

If a view name is specified, only the specified view name is processed for all cells specified in the t_cellName argument. If a view name is not specified, all views of each cell are processed, and index.pcl and cache.pcl files are updated.

Value Returned

t	Version cache has been disabled.
nil	Version cache has not been disabled.

Example

The following code snippet shows how to disable version cache on all instances of the device reflib/FGR2 in all layout cellviews in the library reflib.

```
devids = ddGetObj("reflib" "FGR2")
lst=list(devids)
vfoGRDisableVersionCache(lst "reflib")
```

vfoGREnableVersionCache

```
vfoGREnableVersionCache(
     l_FGRcellDDID
     t_libName
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     )
     => t / nil
```

Description

Enables version cache on all instances currently not cache-enabled that belong to the specified list of FGR super-masters and found in the in the specified library, cell, or view. The `createVersionCache` parameter of such instances will get updated from `0` to the VFO version of the current Virtuoso session.

Note: The function `vfoGREnableVersionCache` calls the `dbSave` function in the background, so any unsaved changes in the layout cellview are saved when the `vfoGREnableVersionCache` function is called.

Arguments

`l_FGRcellDDID` The list of `dd IDs`, returned by the `ddGetObj()` function, for the fluid guard ring cells to be updated and cache-enabled.

`t_libName` The library name for which you want to enable version cache on the FGR instances with latest `vfoVersionCache` values.

`?cellName` *`t_cellName`*

If a cell name is specifed, only cells with the given name are processed and all others are ignored. All views for the given cell name are processed, and `cache.pcl` and `index.pcl` files are updated in all views of the specified cell name. If no cell name is specified, all cells are processed, and `cache.pcl` and `index.pcl` are updated.

`?viewName` *`t_viewName`*

If a view name is specified, only the specified view name is processed for all cells specified in the `t_cellName` argument. If a view name is not specified, all views of each cell are processed, and `index.pcl` and `cache.pcl` files are updated.

Value Returned

`t`	Version cache has been enabled.
`nil`	Version cache has not been enabled.

Example

The following code snippet shows how to enable version cache for all the instances of the device `reflib/FGR2` in all cellviews named as `layout2` of all the cells in the library `reflib`.

```
devids = ddGetObj("reflib" "FGR2")
lst=list(devids)
vfoGREnableVersionCache(lst "reflib" ?viewName "layout2")
```

vfoGRGetCreateFormFieldProp

```
vfoGRGetCreateFormFieldProp(
     t_promptName
     t_propertyName
     )
     => value
```

Description

Gets the value of a property assigned to an existing field or GUI component displayed on the Create Guard Ring form.

Arguments

t_promptName	The field or GUI component name as it appears on the Create Guard Ring form, such as *Device* or *Technology*.
t_propertyName	The property of the specified field. The following values are acceptable for this argument: `invisible`, `enabled`, `editable`, `defValue`, and `value`.

Value Returned

value	The value of the specified property of the given field or GUI component.

Example

```
vfoGRGetCreateFormFieldProp("Contact Rows" "invisible")
    => nil
```

vfoGRGetCreateFormIdentifier

```
vfoGRGetCreateFormIdentifier(
     r_formPointer
     )
     => t_uniqueFormIdentifier
```

Description

Returns the unique identifier string used for creating the Create Guard Ring form.

Arguments

r_formPointer The form pointer created using vfoGRNewCreateForm.

Value Returned

t_uniqueFormIdentifier

A unique identifier used for creating the Create Guard Ring form.

Example

Say, you created the form pointer, `MODGEN`, using the following:

```
form_pointer = vfoGRNewCreateForm("MODGEN" 'OKCancelApply)
```

Then, when you provide the following:

```
uniqueStr = vfoGRGetCreateFormIdentfier(form_pointer)
```

the return value of `uniqueStr` will be "`MODGEN`".

vfoGRGetCreateFormPointer

```
vfoGRGetCreateFormPointer(
     t_uniqueFormIdentifier
     )
     => r_formPointer
```

Description

Returns the form pointer corresponding to the unique identifier that creates the Create Guard Ring form.

Arguments

t_uniqueFormIdentifier

A unique identifier used for creating the Create Guard Ring form, which can be retrieved using vfoGRGetCreateFormIdentifier.

Value Returned

r_formPointer The form pointer created using vfoGRNewCreateForm.

Example

```
errset(let(()
    ;Set a new Create Guard Ring form pointer.
    formStringId = "modgen"

    ;Provide the following to get the form pointer.
    modgenFGRForm = vfoGRGetCreateFormPointer(formStringId)

    ;If the form pointer does not exist, create a new Create Guard Ring form pointer
    using the vfoGRNewCreateForm SKILL function with the provided callbackList.
    unless(modgenFGRForm
        modgenFGRForm = vfoGRNewCreateForm(formStringId 'OKCancelApply
        list("_mgFormOKCB(modgenFGRForm formStringId modgenid)" ""))

        ;Then register the callback for the 'modgenFGRForm' form pointer to call
        the customized form.
        vfoGRRegCreateFormUpdateCallback(modgenFGRForm
             "_mgFormCustomizeCB")
```

```
    )

    ;Display the form on screen.
    when(modgenFGRForm
        hiDisplayForm(modgenFGRForm))
));
```

vfoGRGetCommonQPtr

```
vfoGRGetCommonQPtr(
     )
     => t_queuePointer
```

Description

Returns the common queue pointer of the Create Guard Ring form customized by you.

In the Create Guard Ring form, there are multiple global lists that enable you to add various form components. The common global list (also called common queue) defines the set of form components that are displayed on all four tabs (*Wrap*, *Path*, *Rect*, and *Polygon*) of this form. For example, the GUI components, such as *Technology*, *Device*, and *Contact Rows*, that are common on all tabs (as shown in the images below) exist in the common queue area. If you add a new component in this area, it will be displayed on all the tabs.

vfoGRAddFieldsInWrapTabQ

vfoGRAddFieldsInPathTabQ

vfoGRAddFieldsInRectTabQ

vfoGRAddFieldsInPolygonTabQ

The following example shows how to add a field into the common queue of the form:

```
;Define a new field named "User Field"
userDefinedField = hiCreateBooleanButton(
    ?name          'userDefinedField
    ?buttonText    "User Field"
    ?buttonLocation 'right
    ?defValue      t
    ?value         t
    ?callback      "userDefinedField_CB()"
)
;Ensure that the user-defined components is added in the vfoGRAddFieldsInCommonQ
list, as shown below
vfoGRAddFieldsInCommonQ=list(
    list(userDefinedField 10:50   180:20  160)
)
```

However, if you want to update the GUI components on only a specific tab of the Create Guard Ring form, use the following lists:

- `vfoGRAddFieldsInPathTabQ` (use for the *Path* tab)
- `vfoGRAddFieldsInRectTabQ` (use for the *Rect* tab)
- `vfoGRAddFieldsInPolygonTabQ` (use for the *Polygon* tab)
- `vfoGRAddFieldsInWrapTabQ` (use for the *Wrap* tab)

Arguments

None

Value Returned

`r_queuePointer` The common queue pointer string.

Example

To set the property value of the newly added form field, `actField`, write the following procedure:

```
procedure( setCreateFormUserFieldProp(actField promptString property value)
let(( newField )
    importSkillVar(vfoGRAddFieldsInCommonQ)
```

```
    evalstring( sprintf( nil "vfoGRGetCommonQPtr()->%s->%s = %s" actField
                ->hiFieldSym property value )
    )
))
```

vfoGRGetExtraArgument

```
vfoGRGetExtraArgument(
     )
     => l_keyValuePair
```

Description

Retrieves the value of user-defined fields, that is, list of '`key value`' pair set using the vfoGRSetExtraArgument SKILL function.

Arguments

None

Value Returned

`l_keyvaluePair` An argument list of '`key value`' pairs.

Example

```
argList = vfoGRGetExtraArgument()
foreach(element argList
   if(car(element) == "mdec" then
       return("t")
   else
       return("nil")
   )
)
```

vfoGRGetExtraArgumentName

```
vfoGRGetExtraArgumentName(
     g_obj
     )
     => l_argName
```

Description

Returns the extra parameter names in a list. The end user should override this function for its class that has been extended from `vfoAdvGuardRing`.

Arguments

g_obj	An instance of the `vfoAdvGuardRing` SKILL++ class or any of its subclasses.

Value Returned

l_argName	The list of extra parameter names.

Example

```
defmethod(vfoGRGetExtraArgumentName((obj vfoAdvGuardRing))
list("userParameter1" "userParameter2")
)
```

vfoGRGetQueuePointer

```
vfoGRGetQueuePointer(
     t_qName
     )
     => t_qPointer
```

Description

Returns the pointer corresponding to the name of the queue of the customized Create Guard Ring form.

In the Create Guard Ring form, there are multiple global lists that enable you to add various form components. The global list (also called queue) defines the set of form components that are displayed on all four tabs (*Wrap*, *Path*, *Rect*, and *Polygon*) of this form.

In addition, the following five queues exist in the Create Guard Ring form where you can add your own fields during form customization: `Common`, `Wrap`, `Path`, `Rect`, and `Polygon`.

Any field added in the common queue is visible in all the tabs of the Create Guard Ring form. To add a GUI component to only a specific tab of the Create Guard Ring form, use one the following lists, as needed:

- `vfoGRAddFieldsInPathTabQ` (use for the *Path* tab)
- `vfoGRAddFieldsInRectTabQ` (use for the *Rect* tab)
- `vfoGRAddFieldsInPolygonTabQ` (use for the *Polygon* tab)
- `vfoGRAddFieldsInWrapTabQ` (use for the *Wrap* tab)
- `vfoGRAddFieldsInCommonQ` (use for making a field visible in all tabs)

Arguments

t_qName	A single string argument that can have any one of the following values: `"commonQ"`, `"wrapTabQ"`, `"pathTabQ"`, `"rectTabQ"`, or `"polygonTabQ"`.

Value Returned

t_qPointer	The queue pointer in form of a string that needs to be evaluated before using.

Example

```
qPtr = evalstring(eval(vfoGRGetQueuePointer("wrapTabQ")))
     qPtr~>grWrapCommon~>value //returns the value of Wrap Common field
     qPtr->grPathWidth->value //returns the value of the Path Width field.
```

Here, `qPtr` has been used to retrieve the value of *Wrap Common* and *Path Width* fields that exist on the *Wrap* tab. It will also get the other properties, such as whether the field is in visible or invisible state, or it is an editable or non-editable field.

vfoGRGeometry

```
vfoGRGeometry(
     (pcell vfoSfPCellClass)
     )
     => t / nil
```

Description

Calls three generic functions in the following order: vfoSfInitialize(pcell), vfoSfDraw(pcell), and then vfoSfFinalize(pcell) for drawing the fluid Pcell geometry.

NOTE: This is the top-level entry point to the `vfo` space-filling Pcell.

Arguments

pcell	The SKILL++ object of class, `vfoSfPcellClass`.

Value Returned

`t`	The fluid Pcell geometry is successfully drawn.
`nil`	The fluid Pcell geometry could not be drawn.

Example

```
; Override draw methods
(defmethod vfoGRGeometry ((pcell vfoGuardRing))
        …
     (vfoSfInitialize pcell)
     (vfoSfDraw pcell)
     (vfoSfFinalize pcell)
    …)
```

vfoGRMaximizeShapes

```
vfoGRMaximizeShapes(
     d_instId
     )
     => t / nil
```

Description

Tries all possible combinations of `fillStyle` and `decompositionMode` to find the combination that maximizes the number of shapes, and thereby the number of contact cuts.

Arguments

d_instId	The instance ID for which the contact cuts need to be maximized.

Value Returnedd

`t`	The contact cuts are maximized.
`nil`	The contact cuts could not be maximized.

Example

```
vfoGRMaximizeShapes(inst)
    => nil
```

vfoGRNewCreateForm

```
vfoGRNewCreateForm(
     t_uniqueFormIdentifier
     t_formType
     [ l_callbackList ]
     )
     => r_formPointer / nil
```

Description

Creates the new Create Guard Ring form, that is, a new form pointer along with the unique fields and form symbol.

Arguments

t_uniqueFormIdentifier

An alphanumeric unique identifier string that will be used for creating the Create Guard Ring form, which can be retrieved using vfoGRGetCreateFormIdentifier.

t_formType

One of the following two values, 'OKCancelApply and 'HideCancelDef, which define the layout of the form.

l_callbackList

List of the callback procedures required for controlling the display of the *OK*, *Cancel*, and *Apply* buttons on the form. This is an optional argument.

If this argument is not provided and you chose the 'HideCancelDef type of form, then you can create an FGR using either of the following modes: *Wrap*, *Path*, *Rect*, and *Polygon*.

However, if you did not set this argument and choose the 'OKCancelApply type of form, then the default callback is provided only for the *Wrap* mode.

Value Returned

r_formPointer	The form pointer associated with the Create Guard Ring form.
nil	The form could not be created.

Example

```
form_pointer = vfoGRNewCreateForm("MODGEN" 'OKCancelApply)
        hiDisplayForm(form_pointer)
```

vfoGRRegCreateFormUpdateCallback

```
vfoGRRegCreateFormUpdateCallback(
     r_formPointer
     t_procedureName
     [ l_callbackList ]
     )
```

Description

Registers the user-defined form customization procedure. This procedure gets called for the particular form pointer after the `vfoGRUpdateCreateForm` trigger is called.

Arguments

r_formPointer	The *formPointer* generated using vfoGRNewCreateForm.
t_procedureName	The name of the user-defined form customization procedure defined for the Create Guard Ring form.
l_callbackList	List of the callback procedures required for controlling the display of the *OK*, *Cancel*, and *Apply* buttons on the form. This is an optional argument.

Value Returned

None

Example

```
; create the new Create Guard Ring form and store its form pointer
form_modgen = vfoGRNewCreateForm ("MODGEN" 'OKCancelApply)
; register the callback for the given form pointer
vfoGRRegCreateFormUpdateCallback (form_modgen "vfoCustomize_modgen")
; define the callback
procedure (vfoCustomize_modgen (formPointer
    ; write your own method body here related
    ; to Create Guard Ring form updates
)
hiDisplayForm(form_modgen)
```

vfoGRSetCreateFormAllFieldsInvisible

```
vfoGRSetCreateFormAllFieldsInvisible(
     { t | nil }
     )
```

Description

Sets the visible property for the predefined system fields, except the *Technology* and *Device* fields, on the Create Guard Ring form. This function cannot be used to set the properties of fields that you add while customizing the Create Guard Ring form.

See Customizing the Create Guard Ring Form for more information.

Arguments

`t`	Specifies that the fields or components should not be visible on the form. Consequently, apart from the *Technology* and *Device* fields on the Create Guard Ring form, all others become invisible.
`nil`	Specifies that all the fields or components should be visible on the form.

Value Returned

None

Example

The following code snippet shows how to hide all pre-defined GUI components except the *Device* and *Technology* fields using the `vfoGRSetCreateFormAllFieldsInVisible` SKILL function when the FGR implementation class different from the default `vfoGuardRing` class:

```
if(vfoGetImplementationClassName(libName  cellName) != "vfoGuardRing" then
    vfoGRSetCreateFormAllFieldsInvisible(t)
    else
        vfoGRSetCreateFormAllFieldsInvisible(nil)
)
```

vfoGRSetCreateFormFieldProp

```
vfoGRSetCreateFormFieldProp(
     t_promptName
     t_propertyName
     t_propertyValue
     )
```

Description

Sets the properties of the fields or components displayed on the Create Guard Ring form.

Arguments

t_promptName The field name as it appears on the Create Guard Ring form.

t_propertyName The property of the specified field. The acceptable values for this argument can be one of the following: `invisible`, `enabled`, `editable`, `defValue`, and `value`.

t_propertyValue

The value of each specified property. Use the following table to identify the types of value that can be specified:

propertyName	**propertyValue**
`invisible`	`t` or `nil`
`enabled`	`t` or `nil`
`editable`	`t` or `nil`
`defValue`	Default value depending on user definition, such as float, Boolean, integer, or string
`value`	Current value depending on user definition, such as float, Boolean, integer, or string

Value Returned

None

Example

```
vfoGRSetCreateFormFieldProp("Contact Rows" "invisible" "nil")
```

vfoGRSetExtraArgument

```
vfoGRSetExtraArgument(
     l_associativeList
     )
```

Description

Makes the values of the user-defined GUI components available for processing to the FGR infrastructure that resides in Virtuoso, and sets the corresponding CDF parameters. This SKILL function helps in binding the front-end data for back-end processing.

Arguments

l_associativeList

An associative list of new fields and their corresponding values. Each element of the list is a '`key value`' pair, where the `key` is the name of the FGR device parameter and `value` is the value associated to it, that is,

```
((<FGR_device_parameter_name> <value_of_GUI_component>)
...)
```

Value Returned

None

Example

```
argList = '(("userDefinedParam" t))
vfoGRSetExtraArgument(argList)
```

vfoGRSmoothen

```
vfoGRSmoothen(
     d_inst
     [ ?tmpLpp t_tmpLpp ]
     [ ?direction t_direction ]
     )
     => d_inst
```

Description

Removes the small notches in the given direction from the fluid shape of the given fluid Pcell instance. The temporary shapes are created on `tmpLpp`.

Arguments

d_inst	The ID of the fluid Pcell instance to smoothen.
?tmpLpp *tmpLpp*	The layer purpose pair to create temporary shapes. The default is ("y0" "drawing").
?direction *direction*	The direction can be one of the following: `in`, `out`, or `least`. The default is `least`.

Value Returned

d_inst	The database ID of the smoothened fluid Pcell instance.

Example

```
vfoGRSmoothen(inst)
    => db:0xf11ec793
```

vfoGRUpdateCreateFormSize

```
vfoGRUpdateCreateFormSize(
     x_width
     x_height
     )
     => t / nil
```

Description

Resizes the Create Guard Ring forms, including the form used for modgens. This function updates the size of all tabs and subtabs proportionately. However, there is no change in the size of the fields on the form.

Note: You should call the vfoGRUpdateCreateFormSize function before the Create Guard Ring form is launched in a Virtuoso session, otherwise the Create Guard Ring form will not be resized. This function can be called in the `.cdsinit` or `libInit.il`.

Arguments

x_width	An integer value that represents the width of the Create Guard Ring form in pixels.
x_height	An integer value that represents the height of the Create Guard Ring form in pixels.

Value Returned

`t`	The Create Guard Ring form was resized.
`nil`	The command was unsuccessful.

Example

```
vfoGRUpdateCreateFormSize(500 700)
```

vfoGRUpdateTunnelLPPs

```
vfoGRUpdateTunnelLPPs (
     )
     => l_lpp / nil
```

Description

Returns the layer-purpose pairs on which tunnelling is allowed. The VFO infrastructure selects common layer-purpose present in the FGR and LPPs returned by the function `vfoGRUpdateTunnelLPPs`. If the function `vfoGRUpdateTunnelLLPs` returns `nil`, all FGR layer-purposes are displayed on the Create Tunnel in Fluid Object form.

Arguments

None

Value Returned

l_lpp	List of layer-purpose pairs on which tunnel is allowed.
`nil`	The command was unsuccessful.

Example

To allow to create tunnel on specified layer-purpose only, define the function that returns layer-purpose list.

```
(procedure vfoGRUpdateTunnelLPPs ( )
    '(("M1" "drawing") ("M2" "drawing") ("V0" "drawing") ("V1" "drawing"))
)
;; Layer purpose present in custom FGR
FGR has layers ("Active" "drawing"), ("Poly" "drawing"), ("M1" "drawing"), ("V0" "drawing"), and ("Nimp" "drawing")
;; Layer purpose present in F3 Tunnel form
("M1" "drawing"), ("V0" "drawing")
```

vfoGRUpdateTunnelOptions

```
vfoGRUpdateTunnelOptions (
     )
     => l_optionTypes / nil
```

Description

Returns the list of supported options for FGR tunnel shape types. VFO infrastructure uses options returned by this function and checks with the existing options. The first value returned by this function is selected by default when the Create Tunnel in Fluid Object form opens. If the `vfoGRUpdateTunnelOptions` function is not defined, returns `nil`, or any invalid option is returned, all tunnel options are displayed on the Create Tunnel in Fluid Object form.

Arguments

None

Value Returned

`l_optionTypes`	List of FGR tunnel option types.
`nil`	The command was unsuccessful.

Example

To allow only path and rectangle type tunnels:

```
;;Define procedure which will return supported options list as
       procedure( vfoGRUpdateTunnelOptions()
             list("Path" "Rectangle")
        )
```

vfoGRUpdateVersionCache

```
vfoGRUpdateVersionCache(
     l_FGRcellDDID
     t_libName
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     )
     => t / nil
```

Description

Updates the `vfoCreateVersion` parameter of an instance to the current Virtuoso version. If the `vfoCreateVersion` parameter is set to `0`, the `vfoGRUpdateVersionCache` function does not update the `vfoCreateVersion` parameter to the latest Virtuoso version. You have to call the `vfoGREnableVersionCache` function to update this parameter to the latest Virtuoso version. If the `vfoCreateVersion` parameter is set to an older Virtuoso version and the image of the sub-master is not found in the cache files, `cache.pcl` and `index.pcl`, the `vfoGRUpdateVersionCache` function does not update the `vfoCreateVersion` parameter to the latest Virtuoso version. You have to call the `vfoGRDisableVersionCache` function and then the `vfoGREnableVersionCache` function to update this parameter to the latest Virtuoso version.

Note: The function `vfoGRUpdateVersionCache` calls the `dbSave` function in the background, so any unsaved changes in the layout cellview are saved when the `vfoGRUpdateVersionCache` function is called.

Arguments

l_FGRcellDDID	The list of dd IDs, returned by the ddGetObj() function, for the fluid guard ring cells to be updated.
t_libName	The library name for which you want to update the vfoVersionCache parameter of the FGR instances with the current Virtuoso version.
?cellName *t_cellName*	If a cell name is specifed, only cells with the given name are processed and all others are ignored. All views for the given cell name are processed, and cache.pcl and index.pcl files are updated in all views of the specified cell name. If no cell name is specified, all cells are processed, and cache.pcl and index.pcl are updated.
?viewName *t_viewName*	If a view name is specified, only the specified view name is processed for all cells specified in the t_cellName argument. If a view name is not specified, all views of each cell are processed, and index.pcl and cache.pcl files are updated.

Value Returned

t	The vfoCreateVersion parameter has been updated.
nil	The vfoCreateVersion parameter has not been updated.

Example

The following code snippet shows how this function updates the cacheCreateVersion parameter to the current version in all the cellviews of the library reflib for instances of the device FGR2.

```
devids = ddGetObj("reflib" "FGR2")
lst=list(devids)
vfoGRUpdateVersionCache(lst "reflib")
```

vfoInstallFluidDeviceFiles

```
vfoInstallFluidDeviceFiles(
     [ ?installPath t_installPath ]
     [ ?overwrite { t | nil } ]
     [ ?showMsg { t | nil } ]
     )
     => t / nil
```

Description

Loads a set of Cadence-supplied SKILL files in a specific sequence in the Virtuoso environment. These files are required to work with fluid guard rings. These files are automatically loaded when you start Virtuoso.

Arguments

`?installPath` *`t_installPath`*

The directory where all the `vfo*.ils` files are stored. By default, Cadence supplies these files at the following location:

```
<install_dir>/tools/dfII/etc/vfo
```

If you do not specify this argument, the function looks for the files in the default location.

`?overwrite { t | nil }`

If the `vfo*.ils` files are loaded, this argument prevents the files from loading again. The default is `nil`. You can set it to `t` to load the files again from a different location. You can specify the path in the *`t_installPath`* argument.

`?showMsg { t | nil }`

Displays messages in CIW as the `vfo*.ils` files load and finally returns the following:

```
Loaded files successfully
t
```

The default is `nil`. If the `?overwrite` argument is set to `t`, the messages display in the CIW irrespective of the `?showMsg` setting.

Value Returned

`t`	The `vfo*.ils` files are loaded successfully.
`nil`	Some or all of the `vfo*.ils` files are not loaded successfully.

Example

```
vfoInstallFluidDeviceFiles()
```

Loads the `vfo*.ils` files from the default location. The function returns `t` after all the files are loaded.

```
vfoInstallFluidDeviceFiles(?installPath "/grid/cic/hier/615/dfII/etc/vfo"
?overwrite t ?showMsg t)
```

Loads the `vfo*.ils` files from the `/grid/cic/hier/615/dfII/etc/vfo` path and displays the file loading messages in the CIW. The function returns `t` after all the files are loaded.

vfoIsCommandInDragMode

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

Description

Returns `t` if the vfo-related command is in drag mode, else returns `nil`.

Arguments

None

Value Returned

`t`	The vfo-related command is in drag mode.
`nil`	The vfo-related command is not in drag mode.

Example

```
vfoIsCommandInDragMode()
```

vfoIsGuardRing

```
vfoIsGuardRing(
     t_object
     )
     => t / nil
```

Description

Returns `t` if the object to be queried is a valid guard ring instance, else returns `nil`.

Arguments

t_object	A valid database object to be queried.

Value Returned

`t`	The object to be queried is a valid guard ring instance.
`nil`	The object to be queried is not a guard ring instance.

Example

```
obj=<FGRInstance>
isFGRInst=vfoIsGuardRing(obj)
```

In this example, `isFGRInst` will be true if the object (`obj`) is a valid FGR instance. Else, the function returns `nil`.

vfoMergeInstances

```
vfoMergeInstances(
     g_obj
     d_inst1
     d_inst2
     [ @rest l_args ]
     )
     => dbId / nil
```

Description

Merges the fluid Pcell instances `inst1` and `inst2`. The *Merge* command for fluid Pcells does not pass these arguments. This function updates the relevant Pcell parameters.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_inst1 *d_inst2*	The database ID of the instances to be merged.
l_args	List of optional arguments that can be passed to the function.

Value Returned

dbId	The database ID of the merged instance, if the `Merge` operation is successful.
`nil`	The instances could not be merged.

Example

```
Obj = vfoAllocateProtocolObj(inst1)
    => stdobj@0x148f5138
vfoMergeInstances(obj inst1 inst2)
    => db:0xf11ec794
```

vfoPostChopCBHandler

```
vfoPostChopCBHandler
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or deregisters a user-defined callback function that is called after the chop command completes on an FGR instance. You can do post-processing on the edited FGR instance in the registered callback function.

The user callback function must be defined before calling the `vfoPostChopCBHandler` function to successfully register it as a callback function.

Arguments

t_mode	Specifies mode to register or deregister a user-defined callback function. Valid values are: ■ `r`: Registers the user-defined callback function. ■ `d`: Deregisters an already registered user-defined callback function.
S_callbackFunction	Name or function symbol of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or deregistered successfully.
`nil`	The user callback function could not be registered or deregistered.

Examples

Registers the user-defined callback function `fixOrigin` for post-chop operation.

```
procedure(fixOrigin(inst)
                print("callback to do any post-processing on the FGR instance")
)
```

```
vfoPostChopCBHandler("r" 'fixOrigin)
or
vfoPostChopCBHandler("r" "fixOrigin")
```

Deregisters the user-defined callback function `fixOrigin` for post-chop operation.

```
procedure(fixOrigin()
                print("callback to do any post-processing on the FGR instance")
)
vfoPostChopCBHandler("d" 'fixOrigin)
or
vfoPostChopCBHandler("d" "fixOrigin")
```

vfoPostReshapeCBHandler

```
vfoPostReshapeCBHandler
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or deregisters a user-defined callback function that is called after the reshape command completes on an FGR instance. You can do post-processing on the edited FGR instance in the registered callback function.

The user callback function must be defined before calling the `vfoPostReshapeCBHandler` function to successfully register it as a callback function.

Arguments

t_mode	Specifies mode to register or deregister a user-defined callback function. Valid values are: ■ `r`: Registers the user-defined callback function. ■ `d`: Deregisters an already registered user-defined callback function.
S_callbackFunction	Name or function symbol of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or deregistered successfully.
`nil`	The user callback function could not be registered or deregistered.

Examples

Registers the user-defined callback function `fixOrigin` for post-reshape operation.

```
procedure(fixOrigin(inst)
                print("callback to do any post-processing on the FGR instance")
)
```

```
vfoPostReshapeCBHandler("r" 'fixOrigin)
or
vfoPostReshapeCBHandler("r" "fixOrigin")
```

Deregisters the user-defined callback function `fixOrigin` for post-reshape operation.

```
procedure(fixOrigin()
                print("callback to do any post-processing on the FGR instance")
)
vfoPostReshapeCBHandler("d" 'fixOrigin)
or
vfoPostReshapeCBHandler("d" "fixOrigin")
```

vfoPostSplitCBHandler

```
vfoPostSplitCBHandler
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or deregisters a user-defined callback function that is called after the split command completes on an FGR instance. You can do post-processing on the edited FGR instance in the registered callback function.

The user callback function must be defined before calling the `vfoPostSplitCBHandler` function to successfully register it as a callback function.

Arguments

t_mode	Specifies mode to register or deregister a user-defined callback function. Valid values are: ■ `r`: Registers the user-defined callback function. ■ `d`: Deregisters an already registered user-defined callback function.
S_callbackFunction	Name or function symbol of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or deregistered successfully.
`nil`	The user callback function could not be registered or deregistered.

Examples

Registers the user-defined callback function `fixOrigin` for post-split operation.

```
procedure(fixOrigin(inst)
                print("callback to do any post-processing on the FGR instance")
)
```

```
vfoPostSplitCBHandler("r" 'fixOrigin)
or
vfoPostSplitCBHandler("r" "fixOrigin")
```

Deregisters the user-defined callback function `fixOrigin` for post-split operation.

```
procedure(fixOrigin()
                print("callback to do any post-processing on the FGR instance")
)
vfoPostSplitCBHandler("d" 'fixOrigin)
or
vfoPostSplitCBHandler("d" "fixOrigin")
```

vfoPostStretchCBHandler

```
vfoPostStretchCBHandler
     t_mode
     S_callbackFunction
     )
     => t / nil
```

Description

Registers or deregisters a user-defined callback function that is called after the stretch command completes on an FGR instance. You can do post-processing on the edited FGR instance in the registered callback function.

The user callback function must be defined before calling the `vfoPostStretchCBHandler` function to successfully register it as a callback function.

Arguments

t_mode	Specifies mode to register or deregister a user-defined callback function. Valid values are: ■ `r`: Registers the user-defined callback function. ■ `d`: Deregisters an already registered user-defined callback function.
S_callbackFunction	Name or function symbol of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or deregistered successfully.
`nil`	The user callback function could not be registered or deregistered.

Examples

Registers the user-defined callback function `fixOrigin` for post-stretch operation.

```
procedure(fixOrigin(inst)
                print("callback to do any post-processing on the FGR instance")
)
```

```
vfoPostStretchCBHandler("r" 'fixOrigin)
or
vfoPostStretchCBHandler("r" "fixOrigin")
```

Deregisters the user-defined callback function `fixOrigin` for post-stretch operation.

```
procedure(fixOrigin()
                print("callback to do any post-processing on the FGR instance")
)
vfoPostStretchCBHandler("d" 'fixOrigin)
or
vfoPostStretchCBHandler("d" "fixOrigin")
```

vfoReInstallAllFluidGuardRingDevices

```
vfoReInstallAllFluidGuardRingDevices(
     t_techFileId
     )
     => t / nil
```

Description

Reinstalls all the FGR devices in the specified technology file.

Arguments

t_techFileId	The ID of the technology file in which the guard ring devices are to be reinstalled.

Value Returned

`t`	The FGR devices are successfully installed.
`nil`	The FGR devices could not be installed.

Example

```
vfoReInstallAllFluidGuardRingDevices(tf2)
```

Reinstalls all the FGR devices in the `tf2` technology file.

vfoReInstallGuardRingDevice

```
vfoReInstallGuardRingDevice(
     t_techFileId
     l_devices
     )
     => t / nil
```

Description

Reinstalls the specified FGR devices in the specified technology file.

Arguments

t_techFileId	The ID of the technology file in which the devices are to be reinstalled.
l_devices	The name of the technology file in which the devices are to be reinstalled.

Value Returned

t	The specified devices are successfully installed.
nil	The specified devices could not be installed.

Example

```
vfoReInstallGuardRingDevice(tf1 list("gr1" "gr2"))
```

Reinstalls the `gr1` and `gr2` FGR devices in the `tf1` technology file.

vfoRotateInstance

```
vfoRotateInstance(
     g_obj
     d_inst1
     x_angle_in_degrees
     [ @rest l_args ]
     )
     => t / nil
```

Description

Applies the rotation angle on the FGR instance. This function applies to Pcells.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_inst1	The database ID of the FGR instances to be rotated.
x_angle_in_degrees	Any angle in degrees as double.
l_args	List of optional arguments that can be passed to the function.

Value Returned

`t`	The rotation angle is applied on the instance successfully.
`nil`	The instance is not rotated.

Example

Before calling the `vfoRotateInstance` function, you should call the `vfoSupportsRotation` function on that instance and check its return value. If the return value is `nil`, any angle rotation is not supported on the instance. If return value is `t`, any angle rotation is supported on the instance. For the supported instances, you can call the `vfoRotateInstance` function to apply any angle rotation.

```
protocolObj = vfoAllocateProtoclObj(instId)
retVal = vfoSupportsRotation(protocolObj instId)
when(retVal
       vfoRotateInstance ( protocolObj instId angle))
```

vfoSetParam

```
vfoSetParam(
     obj
     s_param
     value
     )
     => value / nil
```

Description

Sets the value of a CDF parameter to the given value and triggers the callback, if any. The callback is triggered recursively so that if the callback changes another parameter, that callback is also triggered, recursively.

Arguments

obj	The object for which the CDF parameter value should be set. The object can be one of the following: ■ instance database object ■ cell CDF ■ instance CDF
s_param	The CDF parameter for which the value is to be set.
value	The value to be set for the CDF parameter.

Value Returned

value	The value of the given parameter. The value may or may not be the value specified in the function depending on what the callback did.
nil	The value could not be returned.

Example

```
vfoSetParam(I1 "xContSpacing" 0.08)
    => 0.08
```

vfoSetParams

```
vfoSetParams(
     d_inst
     l_paramlist
     )
     => t / nil
```

Description

Sets the value of multiple parameters in a single call. The *l_paramlist* argument contains a list of parameter name and value pairs.

Arguments

d_inst	The ID of the fluid Pcell instance for which the parameter values are to be set.
l_paramlist	The list of parameter name and value pairs to be set. The *l_paramlist* argument should be specified as a list: `(list (<s_param1> <value>) list (<s_param2> <value>)...)`

Value Returned

`t`	The values of the parameters are successfully set.
`nil`	The values of the parameters could not be set.

Example

```
vfoSetParams(I1 '(("xContSpacing" 0.1) ("xContWidth" 0.12)))
    => t
```

vfoSetProtocolClassName

```
vfoSetProtocolClassName(
     cellview
     className
     )
     => t / nil
```

Description

Sets the protocol class for fluid Pcell for the given Pcell supermaster. This function gets called in the Pcell code for creating the supermaster of a fluid guard ring device in a text file.

Arguments

cellview	The Pcell supermaster cellview.
className	The symbol of the protocol class for the fluid Pcell.

Value Returned

t	The protocol class for the fluid Pcell was successfully set.
nil	The protocol class for the fluid Pcell could not be set.

Example

- If the Pcell `vfoProtocolClass` parameter is set to `vfoSfImplClass`, then:

```
vfoSetProtocolClassName (pcell~>cv     pcell~>vfoProtocolClass)
    => vfoSfImplClass
```

- Sets the Pcell `vfoProtocolClass` parameter as `myProtocolClass` while defining the cellview:

```
vfoSetProtocolClassName (tcCellView 'myProtocolClass)
    => myProtocolClass
```

vfoSfDraw

```
vfoSfDraw(
     (pcell vfoSfPCellClass)
     )
     => t / nil
```

Description

Draws the Pcell geometries using the information populated by the `vfoSfInitialize` function.

Arguments

pcell	SKILL++ object of `vfoSfPcellClass`.

Value Returned

`t`	The Pcell geometries are successfully drawn.
`nil`	The Pcell geometries could not be drawn.

Example

```
vfoSfDraw(obj)
    => t
```

vfoSfFinalize

```
vfoSfFinalize(
     pcell vfoSfPCellClass
     )
     => t / nil
```

Description

Performs any required cleanup.

Arguments

pcell	SKILL++ object of `vfoSfPcellClass`.

Value Returned

`t`	The cleanup was performed successfully.
`nil`	The cleanup could not be performed.

Example

```
vfoSfFinalize(obj)
    => nil
```

vfoSfInitialize

```
vfoSfInitialize(
     (pcell vfoSfPCellClass)
     )
     => t / nil
```

Description

This is the initialization function that is called during space-filling Pcell evaluation. This function initializes the slots and populates other required information for Pcell drawing (`vfoSfDraw`).

Arguments

pcell	SKILL++ object of `vfoSfPcellClass`.

Value Returned

`t`	The initialization was successful.
`nil`	The initialization was not successful.

Example

```
vfoSfInitialize(obj)
    => t
```

vfoSfOuterEdgeCornerContRemoval

```
vfoSfOuterEdgeCornerContRemoval(
     (pcell vfoSfPCellClass)
     )
     => t / nil
```

Description

Removes the ear-vertex and end-corner contacts from the corners of an FGR to ensure it is created with a smaller footprint. This is achieved by allowing specification of enclosure values lesser than the maximum value, but greater than the minimum value of the `minOppExtension` rule.

Arguments

pcell	SKILL++ object of `vfoSfPcellClass`.

Value Returned

`t`	The removal of contacts from the fluid guard ring corner was successful.
`nil`	The removal of contacts from the fluid guard ring corner was unsuccessful.

Example

```
vfoSfOuterEdgeCornerContRemoval(pcell)
    => t
```

vfoSmoothen

```
vfoSmoothen(
     d_shape
     s_direction
     [ ?tmpLpp t_tmpLpp ]
     )
     => t / nil
```

Description

Smoothens the given polygon shape. If the shape is not a polygon, then the shape is returned unchanged.

Arguments

d_shape	The ID of the shape to smoothen.
s_direction	The direction can be `in`, `out`, or `least`. The default is `least`.
`?tmpLpp` *t_tmpLpp*	The layer purpose pair to create temporary shapes. The default is '("`y0`" "`drawing`").

Value Returned

`t`	The polygon shape was smoothened.
`nil`	The polygon shape cannot be smoothened.

Example

```
vfoSmoothen(Shape1)
    => t
```

vfoSupportsAbut?

```
vfoSupportsAbut?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the `Abut` operation is supported on the given instance.

Arguments

g_obj	An instance of SKILL++ class `vfoAbstractClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The `Abut` operation is supported on the given instance.
`nil`	The `Abut` operation is not supported on the given instance.

Example

```
obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsAbut?(obj inst)
    => t
```

vfoSupportsChop?

```
vfoSupportsChop?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the `Chop` operation is supported on the given instance.

Arguments

g_obj	An instance of SKILL++ class `vfoAbstractClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The `Chop` operation is supported on the given instance.
`nil`	The `Chop` operation is not supported on the given instance.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsChop?(obj inst)
    => t
```

vfoSupportsConvertToPolygon?

```
vfoSupportsConvertToPolygon?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the `Convert to Polygon` operation is supported on the given instance.

Arguments

g_obj	An instance of SKILL++ class `vfoAbstractClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The `Convert to Polygon` operation is supported on the given instance.
`nil`	The `Convert to Polygon` operation is not supported on the given instance.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsConvertToPolygon?(obj inst)
    => t
```

vfoSupportsCreateObstruction?

```
vfoSupportsCreateObstruction?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the *Create Obstruction* (Tunnel) operation is supported on the given instance.

Arguments

g_obj	An instance of SKILL++ class `vfoAbstractClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The *Create Obstruction* (Tunnel) operation is supported on the given instance.
`nil`	The *Create Obstruction* (Tunnel) operation is not supported on the given instance.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsCreateObstruction?(obj inst)
    => t
```

vfoSupportsCreateObstructions?

```
vfoSupportsCreateObstructions?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the *Create Obstruction* (Tunnel) operation is supported on the given instance.

Arguments

g_obj	An instance of SKILL++ class `vfoSfImplEditClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The *Create Obstruction* (Tunnel) operation is supported on the given instance.
`nil`	The *Create Obstruction* (Tunnel) operation is not supported on the given instance.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsCreateObstructions?(obj inst)
    => t
```

vfoSupportsDeleteObstruction?

```
vfoSupportsDeleteObstruction?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the *Delete Obstruction* (Heal) operation is supported on the given instance.

Arguments

g_obj	An instance of SKILL++ class `vfoAbstractClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The *Delete Obstruction* (Heal) operation is supported on the given instance.
`nil`	The *Delete Obstruction* (Heal) operation is not supported on the given instance.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsDeleteObstruction?(obj inst)
    => t
```

vfoSupportsMerge?

```
vfoSupportsMerge?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the *Merge* operation is supported on the given instance.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The `Merge` operation is supported on the given instance.
`nil`	The `Merge` operation is not supported on the given instance.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsMerge?(obj inst)
    => t
```

vfoSupportsRotation

```
vfoSupportsRotation(
     d_instId
     )
     => t / nil
```

Description

Checks of the Pcell instance supports rotation.

Arguments

d_instId	The instance ID on which the operation is invoked. The Pcell instance ID of `vfoAbstractClass` SKILL++ class or any of its subclasses.

Value Returned

`t`	The `Rotation` operation is supported on the given instance.
`nil`	The `Rotation` operation is not supported on the given instance.

Example

```
protocolObj = vfoAllocateProtoclObj(instId)
retVal = vfoSupportsRotation(protocolObj instId)
if(retVal
   printf("instance supports rotation")
   printf("instance does not supports rotation")
)
```

vfoSupportsUpdateModelShape?

```
vfoSupportsUpdateModelShape?(
     g_obj
     d_instId
     )
     => t / nil
```

Description

Checks if the fluid shape of the given instance can be modified. This function is called by commands that directly update the fluid shape, such as `Stretch`, `Reshape`, `Split`, and `Align`.

Arguments

g_obj	An instance of the `vfoAbstractClass` SKILL++ class or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.

Value Returned

`t`	The fluid shape of the given instance can be modified.
`nil`	The fluid shape of the given instance cannot be modified.

Example

```
Obj = vfoAllocateProtocolObj(inst)
    => stdobj@0x148f5138
vfoSupportsUpdateModelShape?(obj inst)
    => t
```

vfoSupportsVersionCache

```
vfoSupportsVersionCache(
     g_obj
     )
     => t / nil
```

Description

Checks whether the *Version Cache Management* feature is supported on the specified instance.

Arguments

g_obj	An instance of the `vfoSfPCellClass` SKILL++ class or any of its subclasses.

Value Returned

`t`	The *Version Cache Management* feature is supported.
`nil`	The *Version Cache Management* feature is not supported.

Example

```
obj=makeInstance('vfoGuardRing)
vfoSupportsVersionCache(obj)
```

vfoTransformFluidShape

```
vfoTransformFluidShape(
     (shapeObj vfoShapeData)
     transform
     )
     => g_instance / nil
```

Description

Applies the transform to the given shapeData. This function modifies the given shapeData object and returns it.

Arguments

shapeObj	SKILL++ object of any subclass of `vfoShapeData`.
transform	The transform operation should be applied.

Value Returned

g_instance	Instance of the protocol class whose name is stored on the `d_obj` Pcell supermaster.
`nil`	Class object is not returned if the protocol class property is not set on the `d_obj` Pcell supermaster.

Example

```
transform ='((10 10) "R0" 1)
    => ((10 10) "R0" 1)
shapeData = vfoCreateShapeData("rect" '(0:0 10:10))
    => stdobj@0x148f51e0
vfoTransformFluidShape(shapeData transform)
    => stdobj@0x1b073050
shapeData~>??
    => (shapeType "rect" bBox
    ((10.0 10.0)
        (20.0 20.0)))
```

vfoTunnelHigherMetalLayers

```
vfoTunnelHigherMetalLayers(
     (pcell vfoSfPCellClass)
     )
     => t / nil
```

Description

Enables tunneling of high metal layers in a stacked metal fluid guard ring device. This function is called during the Pcell evaluation of fluid guard ring devices.

Arguments

pcell	SKILL++ object of `vfoSfPcellClass`.

Value Returned

`t`	The tunneling of high metal layers was successful.
`nil`	The tunneling of high metal layers was not successful.

Example

```
vfoTunnelHigherMetalLayers(obj)
t
```

vfoUpdateModelShape

```
vfoUpdateModelShape(
     g_obj
     d_instId
     shapeId
     newShapeType
     newPointList
     [ @rest l_args ]
     )
     => t / nil
```

Description

Updates the given fluid shape of the Pcell instance. The new shape type and points are specified in the `newShapeType` and `newPointList` arguments, respectively. The `newPointList` points are in the design coordinates. This function transforms the `newPointList` before updating the fluid shape. The `Edit` command for fluid shapes does not pass these arguments. This function updates the relevant Pcell parameters.

Arguments

g_obj	An instance of SKILL++ class `vfoAbstractClass` or any of its subclasses.
d_instId	The instance ID on which the operation is invoked.
shapeId	The ID of the fluid shape. This argument is useful when an instance contains multiple fluid shapes.
newShapeType	The new shape type for the fluid shape.
newPointList	The new points for the fluid shape.
l_args	List of optional arguments that can be passed to the function.

Value Returned

`t`	The fluid shape of the given instance is successfully updated.
`nil`	The fluid shape of the given instance could not be updated.

Example

```
vfoGetParam(inst "shapeType")
    => "polygon"
vfoUpdateModelShape(obj inst Shape1 "rect" list((2 2) (3 1)))
    => t
vfoGetParam(inst "shapeType")
    => "rect"
```

xfgrAddOption

```
xfgrAddOption(
     ?optionName g_optionName
     ?optionLabel g_optionLabel
     ?optionType g_optionType
     ?optionWidgetType g_optionWidgetType
     ?optionDefaultValue g_optionDefaultValue
     ?supportedOptionValue g_supportedOptionValue
     ?groupName g_groupName
     ?deviceName g_deviceName
     )
     => t / nil
```

Description

Adds a new field as specified in the option name to the Create Fluid Guard Ring form.

Arguments

`?optionName` *`g_optionName`*

The name of the option name to be added to the Create Fluid Guard Ring form.

`?optionLabel` *`g_optionLabel`*

The label of the option added to the Create Fluid Guard Ring form.

`?optionType` *`g_optionType`*

The type of option.
Valid values: `boolean`, `string`, `integer`, `double`.

`?optionWidgetType` *`g_optionWidgetType`*

The type of GUI widget for the specified option name.
Valid values: `none`, `check box`, `line edit`, `combobox`, `spinbox`, or `radiobutton`.

`?optionDefaultValue` *`g_optionDefaultValue`*

The default value for the option.

`?supportedOptionValue` *`g_supportedOptionValue`*

The values supported for the option. This is applicable for widget types `combobox`, `spinbox`, and `radiobutton`.

`?groupName` *`g_groupName`*

The location of the option on form.
Default value: `CustomizedOptions`

`?deviceName` *`g_deviceName`*

The device name for which this option is valid. By default, the option is applicable for all devices.

Value Returned

`t`	The option was added to the Create Fluid Guard Ring form.
`nil`	The option was not added to the Create Fluid Guard Ring form.

Example

Adds the new field, `testOptionComboBox`, to the Create Fluid Guard Ring form.

```
if(xfgrAddOption(?optionName "testOptionComboBox" ?optionLabel "ComboBox Label"
?optionType "string" ?optionWidgetType "combobox"  ?groupName "CustomizedOptions"
?optionDefaultValue "option1" ?supportedOptionValue '("option1" "option2"
"option3"))
    info("Option added successfully")
else
    info("Option not added")
```

xfgrAddRow

```
xfgrAddRow(
     t_tableName
     [ l_values ]
     )
     => t / nil
```

Description

Adds a row with the specified values in a table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
l_values	List of row values.

Value Returned

t	The function is successful**.**
nil	The function is not successful**.**

Example

Adds a row in the specified table.

```
procedure(fgrCreateFormInit()
let((colProps rowProps)
    ;;Adding table for implant layer refernce
    colProps = list(list("layer1" "string" "Layer1")
        list("layer2" "string" "Layer2")
        list("spacing" "double" "Spacing" "lineedit" 0.0)
    )
    xfgrAddTable(?tableName "myTable" ?tableColProps colProps )
))

procedure(optionValueChanged(optionName value prevValue)
    xfgrAddRow("myTable" list("Metal1" "Oxide" "0.02"))
    )
;; step 1
xfgrConfigureOptionsCB("r" "fgrCreateFormInit")

;; step 2
xfgrOptionValueChangeCB("r" "optionValueChanged")
```

xfgrAddTable

```
xfgrAddTable(
     t_tableName
     [ ?tableColProps l_tableColProps ]
     [ ?tableRowProps l_tableRowProps |
          ?tableRowDefaultValues l_tableRowDefaultValues ]
     [ ?groupName t_groupName ]
     [ ?deviceName t_deviceName ]
     [ ?defaultRows x_defaultRows ]
     [ ?minRows x_minRows ]
     [ ?maxRows x_maxRows ]
     )
     => t / nil
```

Description

Adds a table to the xFGR Create form.

Arguments

t_tableName The name of the table.

?tableColProps *l_tableColProps*

List of column and the properties of the column. Supported column types are integer, double, string, radio button, spin box, combo box, and check box.

?tableRowProps *l_tableRowProps* | ?tableRowDefaultValues *l_tableRowDefaultValues*

- ?tableRowProps: List of the values for rows. Each entry should have elements equal to column number and its value should match with type of column. This argument fixes the rows in table. You cannot add or delete rows.
- ?tableRowDefaultValues: The default row value. You can add or delete rows.

?groupName *t_groupName*

The name of group where the table is be added. The group is the section in the xFGR Create form where custom fields will be added. Default value: Custom Options

?deviceName *t_deviceName*

The name of fluid guard ring devices for which the table is visible. By default, the table is visible for all fluid guard ring devices.

?defaultRows *x_defaultRows*

The number rows available when the table is visible for the first time. This argument is only applicable when the argument tableRowDefaultValues has been specified.

?minRows *x_minRows*

The minimum number of rows in the table. The default value is 0. This argument is only applicable when argument the ?tableRowDefaultValues has been specified.

?maxRows *x_maxRows*

The maximum number of rows in the table. The default value is 9999. This argument is only applicable when the argument ?tableRowDefaultValues has been specified.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

Example

Adds a table to the xFGR Create form.

```
procedure(_xfgrCreateFormInit()
let((colProps rowProps)
    ;;There will be 4 columns. The type will be string, string, double, double
    colProps = list(list("layer1" "string" "Layer1")
                list("layer2" "string" "Layer2")
                list("spacing" "double" "Spacing" "lineedit" 0.0)
                list("layer2Width" "double"  "Layer2 Width" "lineedit" 0.0)
            )
    rowProps = list(list("" "" 0.0 0.0) list("" "" 0.0 0.0)) ;;This will avoid add/
delete push button on table
    xfgrAddTable(?tableName "implant_layer_ref" ?groupName "Implant Layers"
?tableColProps colProps ?tableRowProps rowProps ?deviceName "testFGR")
))
xfgrConfigureOptionsCB("r" "_xfgrCreateFormInit")
```

xfgrConfigureOptionsCB

```
xfgrConfigureOptionsCB(
     t_mode
     s_callbackFunction
     )
     => t / nil
```

Description

Registers or de-registers the specified user-defined callback function against `xfgrConfigureOptions`.

Arguments

t_mode	The function mode. Valid values: `r` for registration and `d` for de-registration.
s_callbackFunction	The name of the user callback function.

Value Returned

`t`	The function is successful.
`nil`	The function is not successful.

Example

Registers the `myInit` function.

```
procedure(myInit()
let(()
xfgrAddOption(?optionName "testOptionStringLineEdit2" ?optionLabel "LineEdit
String Label" ?optionType "string" ?optionWidgetType  "lineedit" ?groupName ""
?optionDefaultValue "fgr")
    )
)
xfgrConfigureOptionsCB("r" "myInit")
```

xfgrDeleteRow

```
xfgrDeleteRow(
     t_tableName
     x_rowNumber
     )
     => t / nil
```

Description

Deletes the specified row from the specified table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
x_rowNumber	The number of the row to be deleted from the table.

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Deleted the second row in the specified table.

```
xfgrDeleteRow("myTable" 2)
```

xfgrDisableOption

```
xfgrDisableOption(
     g_optionName
     )
     => t / nil
```

Description

Disables the specified option on the Create Fluid Guard Ring form.

Arguments

g_optionName	The name of an existing fluid guard ring option or a custom option name that has been added to the form.

Value Returned

`t`	The specified option has been disabled on the Create Fluid Guard Ring form.
`nil`	The specified option has not been disabled on the Create Fluid Guard Ring form.

Example

Disables the `wrapType` option on the Create Fluid Guard Ring form.

```
if(xfgrDisableOption("wrapType") then
    info("wrapType option is disabled")
else
    info("wrapType option does not exist")
)
```

xfgrEnableOption

```
xfgrEnableOption(
     g_optionName
     )
     => t / nil
```

Description

Enables the specified option on the Create Fluid Guard Ring form.

Arguments

g_optionName	The name of an existing fluid guard ring option or a custom option name that has been added. to the form.

Value Returned

`t`	The specified option has been enabled on the Create Fluid Guard Ring form.
`nil`	The specified option has not been enabled on the Create Fluid Guard Ring form.

Example

Enables the `wrapType` option on the Create Fluid Guard Ring form.

```
if(xfgrEnableOption("wrapType") then
    info("wrapType option is enabled")
else
    info("wrapType option does not exist")
)
```

xfgrGetAvailableValues

```
xfgrGetAvailableValues(
     g_optionName
     )
     => t / nil
```

Description

Returns the available values for the specified option name. The options are combo box, spin box, or radio button.

Arguments

g_optionName	The name of an existing fluid guard ring option or a custom option name that have been defined and added to the form.

Value Returned

`t`	The available values are returned.
`nil`	The available values are not returned.

Example

Returns the available values of the `optionName`.

```
info("Available values for option %L are %L" optionName
xfgrGetAvailableValues(optionName)
```

xfgrGetCellProps

```
xfgrGetCellProps(
     t_tableName
     x_rowNumber
     t_colName
     [ l_propertiesName ]
     )
     => l_propertiesValue / nil
```

Description

Returns the properties of a cell in the specified table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
x_rowNumber	The number of the row to be deleted from the table.
t_colName	The name of a column of the table.
l_propertiesName	The list of the name of properties. The valid properties are `enable`, `value`, `bgColor`, and `textColor`.

Value Returned

l_propertiesValue	The value of the properties in the sequence they were specified.
`nil`	The function is not successful.

Example

Returns the properties of a cell in the specified table.

```
procedure(tableValueChanged(tableName row colName value prevValue)
    xfgrGetCellProps(tableName row colName list("enable" "textColor" "bgColor"))
)
;;xFGR Create Form customization callback. It is triggered when cell is updated.
xfgrTableCellValueChangedCB("r" "tableValueChanged")
```

xfgrGetColumnProps

```
xfgrGetColumnProps(
     t_tableName
     t_colName
     [ l_propertiesName ]
     )
     => l_propertiesValue / nil
```

Description

Returns the properties of a column of the specified table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
t_colName	The name of a column of the table.
l_propertiesName	The list of the name of properties. The valid properties are `visible`, `enable`, `defValues`, `validValues`, and `label`.

Value Returned

l_propertiesValue	The value of the properties in the sequence they were specified.
`nil`	The function is not successful**.**

Example

Returns the properties of a column of the specified table.

```
procedure(tableValueChanged(tableName row colName value prevValue)
    when(tableName == "myTable" && colName == "myColumn"
        xfgrGetColumnProps("myTable" "myColumn" list("visible" "label" "enable")
    )
)

;;xFGR Create Form customization callback. It is triggered when cell is updated.
xfgrTableCellValueChangedCB("r" "tableValueChanged")
```

xfgrGetFormSize

```
xfgrGetFormSize(
     )
     => l_size / nil
```

Description

Returns the size of the xFGR Create form.This function is used from the xFGR Create form customization callback functions.

Arguments

None

Value Returned

l_size	The size of the xFGR Create form.
nil	The function is not successful**.**

Example

Returns size of the xFGR Create form.

```
procedure(fgrCreateFormInit()
    let((formSize)
formSize = xfgrGetFormSize()
            when(formSize
                xfgrSetFormSize(car(formSize)+50 cadr(formSize)+50)
            )
        )
)
xfgrConfigureOptionsCB("r" "fgrCreateFormInit")
```

xfgrGetGroupProps

```
xfgrGetGroupProps(
     t_groupName
     l_propertiesName
     )
     => l_value / nil
```

Description

Returns the value of the specified property of a PDW in the xFGR Create form. This function is used from the xFGR Create form customization callback functions.

Arguments

t_groupName	The name of the group.
l_propertiesName	The list of name of the properties.

Value Returned

l_value	The list of values of the specified property.
nil	The function is not successful**.**

Example

Returns the value of the specified property.

```
procedure(fgrCreateFormInit()
    let(()
        when(xfgrGetGroupProps ("Implant Layer Settings" "visible")
        xfgrGetGroupProps ("Implant Layer Settings" ?visible nil)
        )
    )
)
xfgrConfigureOptionsCB("r" "fgrCreateFormInit")
```

xfgrGetOptionProps

```
xfgrGetOptionProps(
     t_optionName
     l_propertiesName
     )
     => t / l_value
```

Description

Returns the value of the specified property or list of properties of an option in the xFGR Create form. This function is used from the xFGR Create form customization callback functions.

Arguments

t_optionName	The name of the option.
l_propertiesName	The list of name of properties.

Value Returned

t	The function is not successful**.**
l_value	The list of values of the specified property.

Example

Returns the value of the specified property.

```
procedure(fgrCreateFormValueChanged(option value prevValue)
    let((props)
        when(option == "device" && xfgrGetOptionProps(option "value") == "testFGR"
                 xfgrSetOptionProps("numrows" ?value 2)
    )
    props =  xfgrGetOptionProps(option list("value" "enable"))
    when(nth(1 props)  == nil
        return(prevValue)
        )
       Return(t)
)
xfgrOptionValueChangeCB("r" "fgrCreateFormValueChanged")
```

xfgrGetOptionValue

```
xfgrGetOptionValue(
     g_optionName
     )
     => t / nil
```

Description

Returns the current value for the specified option.

Arguments

g_optionName	The name of an existing fluid guard ring option or a custom option name that has been added to the form.

Value Returned

`t`	The value of the specified option is returned.
`nil`	The value of the specified option is not returned.

Example

Returns the current value, `encloseBy`, for the `wrapType` option.

```
info("The current value of option wrapType is %L"
xfgrGetOptionValue("wrapType" "encloseBy"))
```

xfgrGetTableProps

```
xfgrGetTableProps(
     t_tableName
     [ l_propertiesName ]
     )
     => l_value / nil
```

Description

Returns the properties of a table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
l_propertiesName	The list of name of properties.

Value Returned

t	The function is not successful**.**
l_value	The list of values of the specified property.

Example

Returns the properties of the specified table.

```
procedure(optionValueChanged(optionName value prevValue)
    xfgrGetTableProps("myTable" list("visible"))
)
;;xFGR Create Form customization callback. It is triggered when any non-table GUI
component changed xfgrOptionValueChangeCB("r" "optionValueChanged")
```

xfgrGetTotalTableRows

```
xfgrGetTotalTableRows(
     t_tableName
     )
     => t_rowNumbers / nil
```

Description

Returns the number of rows in the specified table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.

Value Returned

t_rowNumbers	The number of rows in the table.
nil	The function is not successful**.**

Example

Returns the number of rows in the specified table.

```
xfgrGetTotalTableRows("myTable")
```

xfgrHideAllGroups

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

Description

Hides all groups in the Create Fluid Guard Ring form.

Arguments

None

Value Returned

`t`	The groups are hidden.
`nil`	The groups are not hidden.

Example

Hides all groups in the Create Fluid Guard Ring form.

```
if(xfgrHideAllOptions() then
    info("Empty form created.")
else
    info("Could not make the form empty")
)
```

xfgrHideAllOptions

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

Description

Hides all current groups in the Create Fluid Guard Ring form.

Arguments

None

Value Returned

`t`	The current groups and options are hidden.
`nil`	The current groups and options are not hidden.

Example

Hides all current groups and options in the Create Fluid Guard Ring form.

```
if(xfgrHideAllOptions() then
    info("Empty form created.")
else
    info("Could not make the form empty")
)
```

xfgrHideGroup

```
xfgrHideGroup(
     g_groupName
     )
     => t / nil
```

Description

Hides the options in the specified group from the Create Fluid Guard Ring form.

Arguments

g_groupName	The name of the group from which the options are to be hidden.

Value Returned

`t`	The group is hidden.
`nil`	The group is not hidden.

Example

Hides the `Wrap Settings` group from the Create Fluid Guard Ring form.

```
if(xfgrHideGroup("Wrap Settings") then
    info("Group Wrap Settings has been hidden")
else
    info("Group Wrap Settings is not available")
)
```

xfgrHideOption

```
xfgrHideOption(
     g_optionName
     )
     => t / nil
```

Description

Hides the specified option from the Create Fluid Guard Ring form.

Arguments

g_optionName	The name of the option to be hidden from the Create Fluid Guard Ring form.

Value Returned

`t`	The option is hidden.
`nil`	The option is not hidden.

Example

Hides the `wrapType` option from the Create Fluid Guard Ring form.

```
if(xfgrHideOption("wrapType") then
    info("wrapType option is hidden")
else
    info("wrapType option does not exist")
)
```

xfgrInitializeOptionsCB

```
xfgrInitializeOptionsCB(
     t_mode
     s_callbackFunction
     )
     => t / nil
```

Description

Registers or de-registers a user-defined callback function to be called during the initialization of option values in the Create Fluid Guard Ring form.

Arguments

t_mode	The function mode. Valid values: `r` for registration and `d` for de-registration.
s_callbackFunction	The name of the user callback function**.**

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Registers the `initOptions` callback function.

```
procedure(initOptions(option value)
    let(()
        xfgrSetOptionValue("wrapType" "encloseBy")
     )
)
xfgrInitializeOptionsCB("r" "initOptions")
```

xfgrOptionValueChangeCB

```
xfgrOptionValueChangeCB(
     t_mode
     s_callbackFunction
     [ g_optionName ]
     )
     => t / nil
```

Description

Registers or de-registers user-defined callback function to be called when the value of any option on the Create Fluid Guard Ring form changes.

Arguments

t_mode	The function mode. Valid values: `r` for registration and `d` for de-registration.
s_callbackFunction	The name of the user callback function**.**
g_optionName	The callback function is called only for the specified option.

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Registers the `optionValueChanged` callback function.

```
procedure(optionValueChanged(option value)
    let(()
         xfgrAddOption(?optionName "testOptionStringLineEdit2" ?optionLabel
"LineEdit String Label" ?optionType "string" ?optionWidgetType  "lineedit"
?groupName "" ?optionDefaultValue "fgr")
    )
)
xfgrOptionValueChangeCB("r" "optionValueChanged")
```

xfgrPostCreateCB

```
xfgrPostCreateCB(
     t_mode
     s_callbackFunction
     )
     => t / nil
```

Description

Registers or de-registers a user-defined callback function after the creation of the fluid guard ring instance.

Arguments

t_mode	The function mode. Valid values: `r` for registration and `d` for de-registration.
s_callbackFunction	The name of the user callback function**.**

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Registers the `fgrInstPostcreation` callback function.

```
xfgrPostCreateCB("r" "fgrInstPostcreation")
procedure(fgrInstPostcreation(fgrInst)
    let(()
        info("Fluid Guard Ring device name  %s" fgrInst->cellName)
    )
)
```

xfgrPreCreateCB

```
xfgrPostCreateCB(
     t_mode
     s_callbackFunction
     )
     => t / nil
```

Description

Registers or de-registers the user-defined callback function before the creation of the fluid guard ring instance.

Arguments

t_mode	The function mode. Valid values: `r` for registration and `d` for de-registration.
s_callbackFunction	The name of the user callback function**.**

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Registers the `devicePrecreation` callback function.

```
procedure(devicePrecreation(args)
    let(()
        info("Creating device with params %L" args)
    )
)
xfgrPreCreateCB("r" "devicePrecreation")
```

xfgrPrintTable

```
xfgrPrintTable(
     t_tableName
     )
     => t / nil
```

Description

Prints the contents of a table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Prints the contents of the specified table.

```
procedure(tableValueChanged(tableName row colName value prevValue)
    when(tableName == "myTable"
    xfgrPrintTable("myTable")
    )
)

;;xFGR Create Form customization callback. It is triggered when cell is updated.
xfgrTableCellValueChangedCB("r" "tableValueChanged")
```

xfgrResetTable

```
xfgrResetTable(
     t_tableName
     )
     => t / nil
```

Description

Resets the contents of a table. This function is used from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.

Value Returned

`t`	The function is successful**.**
`nil`	The function is not successful**.**

Example

Resets the contents of the specified table.

```
procedure(tableValueChanged(tableName row colName value prevValue)
    when(tableName == "myTable"
        xfgrResetTable("myTable")
    )
)
;;xFGR Create Form customization callback. It is triggered when cell is updated.
xfgrTableCellValueChangedCB("r" "tableValueChanged")
```

xfgrSetAvailableValues

```
xfgrSetAvailableValues(
     g_optionName
     g_supportedOptionValue
     )
     => t / nil
```

Description

Specified the list of valid values for the specified option.

Arguments

g_optionName	The name of the option.
g_supportedOptionValue	The values supported for the option.

Value Returned

t	The supported values have been specified.
nil	The supported values have not been specified**.**

Example

Sets `Place at Minimum Distance` and `Enclose By` as the supported values for the option, `wrapType`.

```
if(xfgrSetAvailableValues("wrapType" '("Place at Minimum Distance" "Enclose By")
then
    info("wrapType possible values can be PlaceAtMinimumDistance or EncloseBy")
else
    info("Available values for option wrapType not updated.")
)
```

xfgrSetCellProps

```
xfgrSetCellProps(
     t_tableName
     x_rowNumber
     t_colName
     [ ?enable {t|nil} ]
     [ ?value t_value ]
     [ ?bgColor t_bgColor ]
     [ ?textColor t_color ]
     )
     => t / nil
```

Description

Sets properties of a cell of a table. The valid properties are `enable`, `value`, `bgColor`, and `textColor`. You can use this function from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
x_rowNumber	Row number of the table.
t_colName	Name of a column of the table.
?enable {t|nil}	Enable or disable a cell. Valid values are: ■ t: Enables a cell of the table. ■ nil: Disables a cell of the table.
?value *t_value*	Value of a cell of the table.
?bgColor *t_bgcolor*	Background color of a cell of the table.
?textColor *t_color*	Color of the text in a cell of the table.

Value Returned

t	The properties of a cell are set.
nil	The properties of a cell are not set.

Example

Sets properties of a cell of a table.

```
procedure(tableValueChanged(tableName row colName value prevValue)
    when(tableName == "myTable"
    xfgrSetCellProps("myTable" row colName ?enable t ?textColor "red" ?bgColor
"white")
    )
    )
;;xFGR Create Form customization callback. It is triggered when cell is updated.
    xfgrTableCellValueChangedCB("r" "tableValueChanged")
```

xfgrSetColumnProps

```
xfgrSetColumnProps (
     t_tableName
     t_colName
     [ ?visible {t|nil} ]
     [ ?enable { t|nil } ]
     [ ?value t_value ]
     [ ?defValue t_defValue ]
     [ ?label t_label ]
     [ ?validValues l_values ]
     )
     => t / nil
```

Description

Sets properties of a column of a table. You can use this function from the xFGR Create form customization callback functions.

Arguments

t_tableName	The name of the table.
t_colName	Name of a column of the table.
?visible {t|nil}	Makes a column visible. Valid values are: ■ t: Makes a column visible. ■ nil: Makes a column invisible.
?enable {t|nil}	Enables or disables a cell. Valid values are: ■ t: Enables a cell of the table. ■ nil: Disables a cell of the table.
?value *t_value*	Value of a cell of the table.
?defValue *t_defValue*	Default value of a column.
?label *t_label*	Label of a column.
?validValues *l_values*	Range of valid values for a combo-box.

Value Returned

t	The properties of a column are set.
nil	The properties of a column are not set.

Example

Sets properties of a column of a table.

```
procedure(tableValueChanged(tableName row colName value prevValue)
    when(tableName == "myTable" && colName == "myColumn"
    xfgrSetColumnProps("myTable" "myColumn" ?visible t ?label "NewLabel")
    )
    )
;;xFGR Create form customization callback. It is triggered when cell is updated.
    xfgrTableCellValueChangedCB("r" "tableValueChanged")
```

xfgrSetFormSize

```
xfgrSetFormSize(
     x_width
     x_length
     )
     => t / nil
```

Description

Sets the size of the xFGR Create form. You can use this function from the xFGR Create form customization callback functions.

Arguments

x_width	The width of the xFGR Create form.
x_length	The length of the xFGR Create form.

Value Returned

`t`	The size of the form is set.
`nil`	The size of the form is not set.

Example

Sets the size of the xFGR Create form.

```
procedure(fgrCreateFormInit()
    let((formSize)
    formSize = xfgrGetFormSize()
        when(formSize
        xfgrSetFormSize(car(formSize)+50 cadr(formSize)+50)
        )
    )
)
xfgrConfigureOptionsCB("r" "fgrCreateFormInit")
```

xfgrSetGroupProps

```
xfgrSetGroupProps(
     t_groupName
     [?visible   g_visibleFlag]
     )
     => t / nil
```

Description

Sets the value of a specified property of a PDW on xFGR Create form. You can use this function from the xFGR Create form customization callback functions.

Arguments

t_groupName	Name of the group.
?visible *g_visibleflag*	Makes a property visible.

Value Returned

t	The function completed successfully.
nil	The function did not complete successfully.

Example

Sets the value of a specified property of a PDW.

```
procedure(fgrCreateFormInit()
    let(()
        xfgrSetGroupProps ("Implant Layer Settings" ?visible nil)
    )
)
xfgrConfigureOptionsCB("r" "fgrCreateFormInit")
```

xfgrSetOptionProps

```
xfgrSetOptionProps(
     t_optionName
     [?visible g_visible]
     [?enable g_enable]
     [?value g_value]
     [?validValues l_validValues]
     )
     => t / nil
```

Description

Sets the properties of an option in the xFGR Create form. You can use this function from the xFGR Create form customization callback functions.

Arguments

t_optionName	The name of the option.
?visible *g_visible*	Makes a column visible. Valid values are: ■ t: Makes a column visible. ■ nil: Makes a column invisible.
?enable *g_enable*	Enables or disables a cell.
?value *g_value*	The value of a cell of the table.
?validValues *l_validValues*	Range of valid values for a combo-box.

Value Returned

t	The function completed successfully.
nil	The function did not complete successfully.

Example

Sets properties of an option in the xFGR Create form.

```
procedure(fgrCreateFormValueChanged(option value prevValue)
    let(()
        if(option == "device" && value == "ngr" then
        xfgrSetOptionProps("numrows" ?value 2)
        else
        xfgrSetOptionProps("numrows" ?value 4)
        )
)
)
xfgrOptionValueChangeCB("r" "fgrCreateFormValueChanged")
```

xfgrSetOptionValue

```
xfgrSetOptionValue(
     g_optionName
     g_optionValue
     )
     => t / nil
```

Description

Sets the specified value for the option.

Arguments

g_optionName	The name of the option.
g_optionValue	The value to be set for the option.

Value Returned

t	The specified value has been set.
nil	The specified values has not been set.

Example

Sets `encloseBy` as the value for the option `wrapType`.

```
if(xfgrSetOptionValue("wrapType" "encloseBy") then
    info("wrapType option value is set to encloseBy")
else
    info("wrapType option doesnot exist")
)
```

xfgrSetTableCellValue

```
xfgrSetTableCellValue(
     t_tableName
     t_rowNumber
     t_columnNumber
     t_value
     )
     => t / nil
```

Description

Sets the value of the specified row and column of the specified table with the specified value in the Create Fluid Guard Ring form.

Arguments

t_tableName	The name of the table in the Create Fluid Guard Ring form or any custom table that has been added to the form.
t_rowNumber	The row number of the cell for which the value needs to be updated.
t_columnNumber	The column number of the cell for which the value needs to be updated.
t_value	The value to be updated in the table cell specified by the row and column number.

Value Returned

`t`	The specified value has been set.
`nil`	The specified value has not been set.

Example

Sets `test` as the value for the cell at row number `1` and column number `3` in the table `Outer Rings`.

```
if(xfgrSetTableCellValue("Outer Rings" "1" "3" "test") then
    info("Value of outerRing 1 netname updated")
else
    info("Value of outerRing 1 netname could not be updated")
)
```

xfgrTableCellValueChangedCB

```
xfgrTableCellValueChangedCB(
     t_registerorDeregister
     t_callbackName
     )
     => t / nil
```

Description

Registers or de-registers a user-defined callback function. The registered callback function is called when a cell of a table is changed.

Arguments

t_registerorDeregister

Registers or de-registers a user-defined callback function. Valid values are:

- `r`: Registers the user-defined callback function.
- `d`: De-registers an already registered user-defined callback function.

t_callbackName Name of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or de-registered successfully.
`nil`	The user callback function could not registered or de-registered.

Example

Registers or de-registers a user-defined callback function

```
;;For registration
    xfgrTableCellValueChangedCB("r" "tableValueChanged")
;;For deregistration
    xfgrTableCellValueChangedCB("d" "tableValueChanged")
```

xfgrSetTableProps

```
xfgrSetTableProps(
     t_tableName
     [ ?visible { t | nil } ]
     )
     => t / nil
```

Description

Sets the properties of a table. You can use this function from the xFGR Create form customization callback functions.

Arguments

t_tablename	The name of the table.
?visible {t|nil}	Makes the property visible. Valid values are: ■ t: Makes a property visible. ■ nil: Makes a property invisible.

Value Returned

t	The property is set.
nil	The property is not set.

Example

Sets the properties of a table.

```
procedure(optionValueChanged(optionName value prevValue)
    xfgrSetTableProps("myTable" ?visible t)
    )
;;xFGR Create form customization callback. It is triggered when any non-table GUI
component changed
    xfgrOptionValueChangeCB("r" "optionValueChanged")
```

xfgrTableRowAddedCB

```
xfgrTableRowAddedCB(
     t_registerorDeregister
     t_callbackName
     )
     => t / nil
```

Description

Registers or de-registers a user-defined callback function. The registered callback function is called when you add a row to a table.

Arguments

t_registerorDeregister

Registers or de-registers a user-defined callback function. Valid values are:

- `r`: Registers the user-defined callback function.
- `d`: De-registers an already registered user-defined callback function.

t_callbackName Name of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or de-registered successfully.
`nil`	The user callback function could not be registered or de-registered.

Example

Registers or de-registers a user-defined callback function.

```
xfgrTableRowAddedCB("r" "tableRowAdded")
```

xfgrTableRowDeletedCB

```
xfgrTableRowDeletedCB(
     t_registerorDeregister
     t_callbackName
     )
     => t / nil
```

Description

Registers or de-registers a user-defined callback function. The registered callback function is executed when you delete a row from a table.

Arguments

t_registerorDeregister

Registers or de-registers a user-defined callback function. Valid values are:

- `r`: Registers the user-defined callback function.
- `d`: De-registers an already registered user-defined callback function.

t_callbackName Name of a user-defined callback function.

Value Returned

`t`	The user callback function was registered or de-registered successfully.
`nil`	The user callback function could not be registered or de-registered.

Example

Registers or de-registers a user-defined callback function.

```
xfgrTableRowDeletedCB("r" "tableRowDeleted")
```

xfgrShowAllGroups

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

Description

Displays all groups in the Create Fluid Guard Ring form.

Arguments

None

Value Returned

`t`	The groups are displayed.
`nil`	The groups are not displayed.

Example

Displays all groups in the Create Fluid Guard Ring form.

```
if(xfgrShowAllGroups() then
    info("All groups are displayed")
else
    info("All/Some groups cannot be displayed")
)
```

xfgrShowGroup

```
xfgrShowGroup(
     g_groupName
     )
     => t / nil
```

Description

Displays the options in the specified group on the Create Fluid Guard Ring form.

Arguments

g_groupName	The name of the group in which the options are to be displayed.

Value Returned

`t`	The group is displayed.
`nil`	The group is not displayed.

Example

Displays the `Wrap Settings` group on the Create Fluid Guard Ring form.

```
if(xfgrShowGroup("Wrap Settings") then
    info("Group Wrap Settings has been displayed")
else
    info("Group Wrap Settings in not available")
)
```

xfgrShowOption

```
xfgrShowOption(
     g_optionName
     )
     => t / nil
```

Description

Displays the specified option on the Create Fluid Guard Ring form.

Arguments

g_optionName	The name of the option to be displayed on the Create Fluid Guard Ring form.

Value Returned

`t`	The option is displayed.
`nil`	The option is not displayed.

Example

Shows the `wrapType` option from the Create Fluid Guard Ring form.

```
if(xfgrShowOption("wrapType") then
    info("wrapType option is visible")
else
    info("wrapType option does not exist")
```

4

Router Translation Functions

This topic provides a list of translator Cadence® SKILL functions associated with the interprocess communication between Virtuoso Studio Design Environment and the router.

These functions are available in mature node releases only.

- iccIsConnected
- iccSendCommand
- iccSendSkillCommand

iccIsConnected

```
iccIsConnected(
     [ d_cellviewId ]
     )
     => t / nil
```

Description

Indicates if a connection from the Virtuoso Studio Design Environment to the router has been established.

Arguments

d_cellviewId	Cellview ID in which the current design is displayed.

Value Returned

t	A connection to the router has been established.
nil	Issues a warning if the design has not been exported.

Related Topics

iccSendCommand

iccSendSkillCommand

iccSendCommand

```
iccSendCommand(
     d_cellviewId
     t_command
     )
     => t_retString
```

Description

Sends a command to the router command line interpreter.

The results are as if you had typed the command in the router command entry field.

Arguments

d_cellviewId	The ID of the cellview.
t_command	A string to pass to the router. Strings that are constants should be quoted. Strings that are variables need not be quoted.

Value Returned

t_retString	Whatever string the router returns after successfully executing a task.

Examples

This example displays the router design report in the report window.

```
iccSendCommand(getEditCellview() "report design")
```

Related Topics

iccIsConnected

iccSendSkillCommand

iccSendSkillCommand

```
iccSendSkillCommand(
     d_cellviewId
     t_command
     )
     => t_retString
```

Description

Sends a SKILL command to the router.

You can use only the core SKILL commands. Other, application-specific commands are not supported in the router.

Arguments

d_cellviewId	The ID of the cellview.
t_command	A SKILL command to pass to the router.

Value Returned

t_retString	Whatever string the router returns after successfully executing a task.

Related Topics

iccIsConnected

5

Custom Digital Placer Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso® Custom Digital Placer.

Only these functions are supported for public use. Any other function, regardless of its name or prefix, and undocumented aspects of the functions described below, are private and subject to change at any time.

- lxDressingTemplateEditor
- lxEditPinPlacement
- lxEditPlacementStyle
- lxHiAutoPlace
- lxHiPlaceBoundaryCell
- lxHiPlaceFiller
- lxMovePorts
- vcpfeCreateRows
- vcpfePlaceBoundaryCells
- vcpfePlaceConstraintGroup
- vcpfePlaceFillers
- vcpfePlaceTapCells
- vcpfeRegisterPostPlacementProc
- vcpfeRunCustomDigitalPlacer

Deleted Custom Digital Placer SKILL Functions

The following Cadence® SKILL functions associated with Virtuoso® Custom Digital Placer are no longer supported in the current release and will be ignored if called. They will be removed completely in the next major release.

■ lxPlcAppendPlaceSetupFieldValue ■ lxPlcGetPlaceSetupFieldChoices ■ lxPlcGetPlaceSetupFieldItems ■ lxPlcGetPlaceSetupFieldValue ■ lxPlcIsPlaceSetupFieldEditable ■ lxPlcIsPlaceSetupFieldEnabled ■ lxPlcReplacePlaceSetupField ■ lxPlcSetPlaceSetupFieldChoices ■ lxPlcSetPlaceSetupFieldEditableState ■ lxPlcSetPlaceSetupFieldEnableState ■ lxPlcSetPlaceSetupFieldItems ■ lxPlcSetPlaceSetupFieldValue ■ lxTpTemplateLoadAll ■ vcpfeCreateEstimatedRows	Use the Placement Planning form and corresponding environment variables to set the default values in the form.
■ vcpPlaceBoundaryCells	This function is no longer supported and will be ignored if specified. It has been retained to maintain backward compatibility.

lxDressingTemplateEditor

```
lxDressingTemplateEditor(
     )
     => t
```

Description

Invokes the Dressing Template Editor form.

Arguments

None

Value Returned

`t`	The Dressing Template Editor form was opened.

Example

Opens the Dressing Template Editor form.

```
lxDressingTemplateEditor()
=> t
```

lxEditPinPlacement

```
lxEditPinPlacement(
     [ g_fromLayoutGenFormP ]
     )
     => t
```

Description

Opens the Pin Placement form in which you can view pin information, constrain a pin to a boundary edge, order and fix pins, specify the pitch for pin placement, and start the *Pin Placement* function.

Arguments

g_fromLayoutGenFormP

Tells the function if the call came from the Generate Layout form. This argument is obsolete and is retained for backward compatibility only.
Valid Values: `t`, `nil`
Default: `nil`

Value Returned

`t` The Pin Placement form was opened.

Example

Opens the Pin Placement form.

```
lxEditPinPlacement()
=> t
```

lxEditPlacementStyle

```
lxEditPlacementStyle(
     )
     => t
```

Description

Opens the Placement Planning form in which you can plan the placement of CMOS, standard cells, or a mixture of CMOS and standard cells. Each of the three modes generates rows based on the parameters you enter in the form.

Arguments

None

Value Returned

`t`	The Placement Planning form was opened.

Example

Opens the Placement Planning form.

```
lxEditPlacementStyle()
=> t
```

lxHiAutoPlace

```
lxHiAutoPlace(
     [ d_cellViewID ]
     )
     => t / nil
```

Description

Opens the Auto Placer form in which you can set placement parameters and run the Virtuoso Custom Digital Placer automatic placement function.

Arguments

d_cellViewID	Database ID of the cellview you want to place.

Value Returned

t	The Auto Placer form was opened.
nil	The Auto Placer form was not opened or the action was canceled.

Example

Opens the Auto Placer form.

```
lxHiAutoPlace(geGetEditCellView())
=> t
```

lxHiPlaceBoundaryCell

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

Description

Opens the Boundary Cell Placement form.

Arguments

None

Value Returned

`t`	The Boundary Cell Placement form was opened.
`nil`	The Boundary Cell Placement form was not opened or the action was canceled.

Example

Opens the Boundary Cell Placement form.

```
lxHiPlaceBoundaryCell()
=> t
```

lxHiPlaceFiller

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

Description

Displays the Insert/Delete Filler Cells form.

Arguments

None

Value Returned

`t`	The Insert/Delete Filler Cells form is displayed.
`nil`	The Insert/Delete Filler Cells form could not be displayed.

Example

Opens the Insert/Delete Filler Cells form.

```
lxHiPlaceFiller()
=> t
```

lxMovePorts

```
lxMovePorts(
     d_cellViewID
     )
     => t
```

Description

Moves top-level ports over components based on their connectivity.

Arguments

d_cellViewID	Database ID of the layout cellview containing top-level ports.

Value Returned

t	The ports were moved over the corresponding components.

Example

Moves top-level ports over components based on their connectivity in the in the specified cellview.

```
lxMovePorts(myCellview1)
=> t
```

vcpfeCreateRows

```
vcpfeCreateRows(
     d_cv
     t_mode
     [ ?partitionName t_partitionName ]
     [ ?area l_area ]
     [ ?createRows g_createRows ]
     [ ?insertTapCells g_insertTapCells ]
     [ ?insertBoundaryCells g_insertBoundaryCells ]
     )
     => t / nil
```

Description

Estimates and creates rows according to the specified arguments. This command also places tap cells and boundary cells after row creation, which can be controlled using arguments.

Arguments

d_cv	The cellview ID.
t_mode	Specifies whether rows must be inserted or deleted. Valid values are: `insert` (default) and `delete`
?partitionName *t_partitionName*	Specifies the region within which rows must be created. The default value is `Boundary`.
?area *l_area*	Specifies the area coordinates within which rows must be generated. The default value is nil, which indicates no area is selected.
?createRows *g_createRows*	Specifies whether existing rows must be replaced with new rows.
?insertTapCells *g_insertTapCells*	Specifies whether tap cells must be inserted in the rows.
?insertBoundaryCells *g_insertBoundaryCells*	Specifies whether boundary cells must be inserted in the rows.

Value Returned

`t`	Rows were generated as per the specifications.
`nil`	The rows were not generated.

Examples

Creates rows and places tap cells.

```
vcpfeCreateRows(cv "insert" ?insertTapCells t)
=> t
```

Creates rows and places boundary cells.

```
vcpfeCreateRows(cv "insert" ?insertBoundaryCells t)
=> t
```

Places tap cells on already created rows.

```
vcpfeCreateRows(cv "insert" ?createRows nil ?insertTapCells t)
=> t
```

Deletes placed boundary cells.

```
vcpfeCreateRows(cv "delete" ?createRows nil ?insertBoundaryCells t)
=> t
```

vcpfePlaceBoundaryCells

```
vcpfePlaceBoundaryCells(
     d_cvId
     t_mode { insert | delete }
     [ ?regionList l_list ]
     [ ?leftCell t_name ]
     [ ?rightCell t_name ]
     [ ?leftBottomCornerCell t_name ]
     [ ?rightBottomCornerCell t_name ]
     [ ?leftTopCornerCell t_name ]
     [ ?rightTopCornerCell t_name ]
     [ ?topEdgeCell t_name ]
     [ ?bottomEdgeCell t_name ]
     [ ?leftMirror { t | nil } ]
     [ ?rightMirror { t | nil } ]
     [ ?leftCornerMirror { t | nil } ]
     [ ?rightCornerMirror { t | nil } ]
     [ ?outside { t | nil } ]
     [ ?topBotRowPlaceable { t | nil } ]
     [ ?boundaryCompTypes t_name ]
     [ ?libraryFilter t_name ]
     [ ?selectedBoundaryCells t_name ]
     )
     => l_cellIDs / nil
```

Description

Places boundary cells around core cells to isolate the core cells from each other.

Arguments

d_cvId Specifies the current cellview ID.

t_mode{insert|delete}

Specifies whether boundary cells are to be inserted or deleted.

?regionList *l_list*

Specifies the row region for which boundary cells must be generated.

?leftCell *t_name*

Specifies the boundary cell to be inserted in the left edge.

?rightCell *t_name*

Specifies the boundary cell to be inserted in the right edge.

?leftBottomCornerCell *t_name*

Specifies the boundary cell to be inserted in the left bottom corner of the row region.

?rightBottomCornerCell *t_name*

Specifies the boundary cell to be inserted in the right bottom corner of the row region.

?leftTopCornerCell *t_name*

Specifies the boundary cell to be inserted in the left top corner of the row region.

?rightTopCornerCell *t_name*

Specifies the boundary cell to be inserted in the right top corner of the row region.

?topEdgeCell *t_name*

Specifies the boundary cell to be inserted in the top edge of the row region.

?bottomEdgeCell *t_name*

Specifies the boundary cell to be inserted in the bottom edge of the row region.

?leftMirror{t|nil}

Specifies whether a mirrored orientation is to be applied to the boundary cells that are present in the left edge of the row region.

`?rightMirror { t | nil }`

Specifies whether a mirrored orientation is to be applied to the boundary cells that are present in the right edge of the row region.

`?leftCornerMirror { t | nil }`

Specifies whether a mirrored orientation is to be applied to the boundary cells that are present in the left corners of the row region.

`?rightCornerMirror { t | nil }`

Specifies whether a mirrored orientation is to be applied to the boundary cells that are present in the right corners of the row region.

`?outside { t | nil }`

Specifies whether boundary cells are to be generated outside the row region. When set to `t`, if the available space between the PR boundary and row region is insufficient for placing boundary cells, the placement region is shrunk by a value equal to the width of the boundary cells. The boundary cells are then generated outside the row region. However, if there is enough space, the boundary cells are placed as specified, without shrinking the placement region.

`?topBotRowPlaceable { t | nil }`

Allow the placement of core cells over boundary cells in the top and bottom rows.

`?boundaryCompTypes` *`t_name`*

Specifies the component types for boundary cells.

`?libraryFilter` *`t_name`*

Specifies the libraries that contain the required boundary cells.

`?selectedBoundaryCells` *`t_name`*

Specifies the cellviews, in the `(lib cell view)` format, that contain the required boundary cells.

Value Returned

l_cellIDs	List of IDs of the boundary cells that were placed.
nil	The boundary cells could not be placed.

Examples

Places boundary cells in the current cellview. The boundary cells for the various edges are specified. The component type of the boundary cells is also specified.

```
vcpfePlaceBoundaryCells(currWindowID "insert" ?regionList geGetSelSet() ?leftCell
"leftCell" ?rightCell "rightCell" ?bottomEdgeCell "botEdge1 botEdge2" ?topEdgeCell
"topEdge1 topEdge2" ?leftBottomCornerCell "lbc" ?leftTopCornerCell "ltc"
?rightBottomCornerCell "rbc" ?rightTopCornerCell "rtc" ?boundaryCompTypes
"bdryCompType")
```

Places boundary cells in the current cellview outside the row region. The boundary cells for the various edges are specified. The `?selectedBoundaryCell` argument defines the boundary cells.

```
vcpfePlaceBoundaryCells(currWindowID "insert" ?regionList geGetSelSet() ?leftCell
"leftCell" ?rightCell "rightCell" ?bottomEdgeCell "botEdge1 botEdge2" ?topEdgeCell
"topEdge1 topEdge2" ?leftBottomCornerCell "lbc" ?leftTopCornerCell "ltc"
?rightBottomCornerCell "rbc" ?rightTopCornerCell "rtc" ?outside t
?selectedBoundaryCell "(lib leftCell layout)(lib rightCell layout) (lib botEdge1
layout)(lib botEdge2 layout) (lib topEdge1 layout)(lib topEdge2 layout) (lib lbc
layout)(lib ltc layout) (lib rbc layout)(lib rtc layout)")
```

vcpfePlaceConstraintGroup

```
vcpfePlaceConstraintGroup(
     d_cellViewID
     )
     => s_constraintGroupName
```

Description

Returns the name of the constraint group that will be used when the placer is run on the specified cellview.

Arguments

d_cellViewID	Database ID of the cellview whose constraint group you want to identify.

Value Returned

s_constraintGroupName

The name of the constraint group in effect for the specified cellview.

Example

Returns the constraint group for the cellview currently being edited.

```
vcpfePlaceConstraintGroup( geGetEditCellView() )
```

vcpfePlaceFillers

```
vcpfePlaceFillers(
     d_cv
     g_mode
     ?partitionName { prBoundary | g_clusterBoundaryName }
     ?fillerCompTypes t_fillerCompTypes
     ?fillerCellNames t_fillerCellNames
     ?ignoreBlockages { t | nil }
     ?createphysOnly { t | nil }
     )
     => t / nil
```

Description

Inserts or deletes filler cells from the specified cellview.

Arguments

d_cv	The cellview ID.
g_mode	Specifies whether the filler cells need to be inserted or deleted. Valid values are `insert` and `delete`.
`?partitionName` { `prBoundary` | *g_clusterBoundaryName* }	Defines the region within which filler cells need to be added or deleted.
`?fillerCompTypes` *t_fillerCompTypes*	Specifies the component types that contain filler cells.
`?fillerCellNames` *t_fillerCellNames*	Specifies the names of the cells that contain filler cells.
`?ignoreBlockages` { `t` | `nil` }	When set to `t`, ignores blockages during filler insertion. Default value is `nil`.
`?createphysOnly` { `t` | `nil` }	When set to `t`, creates physOnly fillers. Default value is `nil`.

Value Returned

`t`	Filler cells were inserted or deleted, depending on the mode selected.
`nil`	Filler cells could not be inserted or deleted.

Examples

`FILL1` and `FILL2` component types are defined as FILLER component class in CPH. `CELL1` and `CELL2` have their `oaCellType` as `coreSpacer` in the design.

```
vcpfePlaceFillers(cv "insert")
```

Inserts fillers in the current cellview with all fillers available.

```
vcpfePlaceFillers(cv "insert" ?fillerCellNames "")
```

Inserts fillers in the current cellview with only `FILL1` and `FILL2` component types.

```
vcpfePlaceFillers(cv "insert" ?fillerCompTypes "FILL1" ?fillerCellNames "")
```

Inserts fillers in the current cellview with only `FILL1` component type.

```
vcpfePlaceFillers(cv "insert" ?fillerCompTypes "" ?fillerCellNames "CELL1")
```

Inserts fillers in the current cellview with only `CELL1` filler cell.

```
vcpfePlaceFillers(cv "insert" ?partitionName "cluster1" ?fillerCompTypes "")
```

Inserts fillers in the current cellview inside a cluster boundary named `cluster1` with `CELL1` and `CELL2` filler cells.

```
vcpfePlaceFillers(cv "insert" ?ignoreBlockages t ?createPhysOnly t)
```

Inserts fillers in the current cellview with all fillers available and ignore blockages. Also create physOnly fillers.

```
vcpfePlaceFillers(cv  "delete")
```

Deletes all fillers in the current cellview.

```
vcpfePlaceFillers(cv "delete" ?fillerCompTypes "FILL1" ?fillerCellNames "")
```

Deletes fillers that are of the `FILL1` component type.

```
vcpfePlaceFillers(cv "delete" ?fillerCompTypes "")
```

Deletes only `CELL1` and `CELL2` fillers.

```
vcpfePlaceFillers(cv "delete" ?partitionName "cluster1")
```

Deletes fillers only from the `cluster1` cluster boundary, and keeps other fillers inside the PR boundary.

vcpfePlaceTapCells

```
vcpfePlaceTapCells(
     d_cv
     t_mode {insert | delete}
     [ ?partitionName t_name ]
     [ ?tapCompTypes t_types ]
     [ ?tapCellNames t_cellNames ]
     [ ?tapToTapSpacing f_value ]
     [ ?tapBackOff f_value ]
     [ ?minTapSpacing f_value ]
     [ ?tapPattern {Regular | Checker Board} ]
     [ ?skipRow x_number ]
     [ ?tapRowOffset f_value ]
     [ ?tapRowEndOffset f_value ]
     [ ?periodicTap {t | nil} ]
     [ ?lockTap {t | nil} ]
     [ ?avoidAbutment {t | nil} ]
     [ ?ignoreBlockages {t | nil} ]
     )
     => l_tapCellIDs / nil
```

Description

Inserts tap cells in the empty spaces between standard cells in the specified cellview as per the specified arguments. When *t_mode* is set to `delete`, all tap cells in the given cellview are deleted.

Arguments

d_cv The cellview ID.

t_mode { insert | delete }

The mode in which the command must be run - whether tap cells must be inserted or deleted.

?partitionName *t_name*

The area in the given cellview, for example a boundary or cluster name, in which tap cells must be inserted or deleted.

Examples: prBoundary (default), name of a row region

?tapCompTypes *t_types*

The component types that contain the required tap cells (also referred as substrate contacts). The specified component types must have the component class `STDSUBCONT`.

?tapCellNames *t_cellNames*

Names of the (one or more) tap cells.

?tapToTapSpacing *f_value*

Gap between adjacent tap cells.

?tapBackOff *f_value*

Minimum distance from the PR boundary or row region boundary to the first or last tap cells. The `?tapRowOffset` and `?tapRowEndOffset` values cannot be lesser than the `?tapBackOff` value.

?minTapSpacing *f_value*

Minimum spacing between adjacent tap cells.

?tapPattern { `Regular` | `Checker Board` }

The pattern in which tap cells must be inserted. Regular pattern is applicable to both single and multi-height tap cells, while `Checker Board` pattern is applicable only for single-height tap cells.

With the `Checker Board` tap pattern selected, the `?skipRow` and `?avoidAbutment` settings are ignored.

?skipRow *x_number*

Skips the specified number of rows while placing tap cells, starting from bottom row. The top row cannot be skipped. The `?skipRow` setting is ignored when `?tapPattern` is set to `Checker Board`.

Examples:

If the value is set to `1`, taps are inserted in rows `1`, `3`, `5`, and so on.

If the value is set to `3`, taps are inserted in rows `1`, `5`, `9`, and so on.

`?tapRowOffset` *`f_value`*

Distance from the left edge of the row at which the first tap cell must be placed in each row.

`?tapRowEndOffset` *`f_value`*

Distance from the right edge of the row at which the last tap cell must be placed in each row.

`?periodicTap` {`t` | `nil`}

When set to `t`, all the available tap cells are inserted periodically starting from bottom lower left corner of the boundary in a circular pattern. The default value is `nil`.

`?lockTap` {`t` | `nil`}

Specifies whether the tap cells must be locked after insertion. The default value is `t`.

`?avoidAbutment` {`t` | `nil`}

Controls the abutment of tap cells. This setting is ignored when `?tapPattern` is set to `Checker Board`. The default value is `nil`, and therefore the tap cells are abutted vertically by default.

`?ignoreBlockages` {`t` | `nil`}

When set to `t`, ignores blockages during tap cell insertion. The default value is `nil`.

Value Returned

l_tapCellIDs	IDs of tap cells that were instantiated or deleted.
nil	The command was unsuccessful.

Examples

Inserts the available tap cells periodically starting from the bottom lower left corner of the boundary.

```
vcpfePlaceTapCells(cv "insert"
   ?tapCompTypes "standsubcont"
   ?tapCellNames  "(acpd MySub layout) (acpd mySub1 layout)"
   ?tapToTapSpacing 0.2
   ?skipRow t
)
```

vcpfeRegisterPostPlacementProc

```
vcpfeRegisterPostPlacementProc(
     s_procName
     )
     => t / nil
```

Description

Registers the specified user-defined post-placement procedure. The procedure is called after running automatic placement.

Arguments

s_procName	Specifies the user-defined procedure to be registered. The signature of the procedure must be in the following format: `myProc(cv instIdList)` Here, `cv` refers to the cellview on which Auto Placer was run and `instIdList` refers to the list of instances placed by Auto Placer.

Value Returned

`t`	The user-defined procedure is registered.
`nil`	The user-defined procedure could not be registered.

Examples

Registers a user-defined procedure called myProc.

```
vcpfeRegisterPostPlacementProc('myProc)
```

Deregisters all post-placement procedures.

```
vcpfeRegisterPostPlacementProc(nil)
```

vcpfeRunCustomDigitalPlacer

```
vcpfeRunCustomDigitalPlacer(
     d_schCvID
     d_layCvID
     [ ?groupCMOSPairs { t | nil } ]
     [ ?preserveChains { t | nil } ]
     [ ?groupMFactors { t | nil } ]
     [ ?allowRotation { t | nil } ]
     [ ?selectedCompList l_selectedCompList ]
     [ ?placementMode { Auto Placement | ECO Placement } ]
     [ ?optimization { none | basic | moderate | optimized } ]
     [ ?ecoModeImpact { none | localized | maximized } ]
     [ ?insertSubstrateContacts { t | nil } ]
     [ ?saveAs { t | nil } ]
     [ ?saveAsLibName t_libName ]
     [ ?saveAsCellName t_cellName ]
     [ ?saveAsViewName t_viewName ]
     [ ?overwrite { t | nil } ]
     [ ?intraRowSpacer { t | nil } ]
     [ ?interRowSpacer t | nil } ]
     [ ?adjustBdy { t | nil } ]
     [ ?reserveTracksForRouting { -1 | 0 | 1 ... 10 } ]
     [ ?vcpRulesConstraintGroup t_constraintGroupName ]
     [ ?minBoundaryOffset f_offset ]
     [ ?maxPinsPerNet x_numPins ]
     [ ?mixedMode { t | nil } ]
     [ ?placeCluster t_clusterName ]
     [ ?componentEdge { Boundary | Bounding Box } ]
     )
     => t
```

Description

Runs the custom digital placer, specifying the type of placement that is done, the technology rules to be used, any automatic spacing adjustments to be done, and how the modified cellview is saved.

Arguments

Each of the arguments listed has an equivalent environment variable. If you do not explicitly set a particular argument, the function defaults to the behavior specified by the corresponding environment variable.

There is no argument that corresponds to the vcpCellBoundaryLPPs environment variable. However, the environment variable setting is taken into account when you run the SKILL function.

d_schCvID { t | nil }

Database ID of the schematic source.

d_layCvID { t | nil }

Database ID of the layout cellview to be placed.

?groupCMOSPairs { t | nil }

Groups pairs of CMOS devices in designs that have been generated with chaining switched on.

This option does nothing for devices that have been abutted manually because clustering needs chaining information.

?preserveChains { t | nil }

Preserves chains that were generated automatically by the *Generate All From Source* or *Placement Planning* commands.

When switched off, the placer is free to break chains in order to share diffusion on individual transistors, which can reduce the wire length of polysilicon gate connections and result in better alignment.

?groupMFactors { t | nil }

Automatically adds a grouping constraint for complementary MOS devices that have a multiplication factor in the schematic.

?allowRotation { t | nil }

Lets the placer rotate components as part of its optimization. When switched off, the placer can move components but not rotate them.

?selectedCompList *l_selectedCompList*

Lists the instIds of components that need to be placed. Default value is nil which means all the components will be placed.

?placementMode { Auto Placement | ECO Placement }

Specifies the mode to be used to place components

Note: Enclose the value in double quotes

`?optimization { none | basic | moderate | optimized }`

Controls the time that the placer spends trying to achieve the best possible results in Auto Placement Mode.

Note: Enclose the value in double quotes

`?ecoModeImpact { none | localized | maximized }`

Specifies the extent to which the placed components within the design can be moved from their original positions in ECO Placement Mode

Note: Enclose the value in double quotes

`?insertSubstrateContacts { t | nil }`

Inserts standard cell substrate contacts in the empty spaces between the standard cells in a row, based on a specified minimum and maximum contact spacing value. You specify the name of the component type that contains the substrate contacts using the substrateContactType environment variable.

Note: This option is honored only when using the *Assisted Standard Cell* and *Assisted Mixed CMOS/Standard-Cell* placement styles.

`?saveAs { t | nil }`

Saves the placed layout under a different cellview name. If the cellview name you specify already exists, you must set `?overwrite` to `t` to enable the placer to run.

The cellview is not saved automatically before placement is run, so be sure to save it to disk before you start in order to preserve any edits that are still in memory. To discard the changes the placer makes, close the output cellview without saving.

`?saveAsLibName` *`t_libName`*

Specifies the library name for the `?saveAs` option.

`?saveAsCellName` *`t_cellName`*

Specifies the cell name for the `?saveAs` option.

`?saveAsViewName` *`t_viewName`*

Specifies the view name for the `?saveAs` option.

`?overwrite { t | nil }`

Automatically overwrites an existing cellview with the same name as that specified by the `?saveAs` options.

If the specified cellview name already exists and `?overwrite` is set to `nil`, the placer does not run.

`?intraRowSpacer { t | nil }`

Adds or removes space between the components within each row.

`?interRowSpacer { t | nil }`

Adds or removes space between each row in order to improve routability. Specify the number of tracks to add between each row using the `?reserveTracksForRouting` option.

`?adjustBdy { t | nil }`

Automatically recalculates the boundary to take into account the spacer options.

`?reserveTracksForRouting { -1 | 0 | 1 ... 10 }`

Specifies the additional number of routing tracks to be reserved between each row when running the `?interRowSpacer` function.

The number you specify is added to the number of tracks the placer estimates are required to perform the routing to give the total number of tracks required. This is then compared to the number of tracks available between rows and the space available increased or reduced accordingly.

For example,

Placer estimates required tracks for routing: 10
reserveTracksForRouting: 3
Total number of tracks required: 13

If there are currently 8 tracks available between rows, the placer adds space for 5 more.
If there are currently 15 tracks available between rows, the placer removes 2.

`?vcpRulesConstraintGroup` *`t_constraintGroupName`*

Specifies the constraint group containing the placement rules for the current technology.

`?minBoundaryOffset` *`f_offset`*

Specifies the minimum spacing allowed between the boundary edge and the nearest component.

The default value is half of the maximum value of all the spacing rules for the layers specified in the applicable technology database.

`?maxPinsPerNet` *`x_numPins`*

Specifies the maximum number of pins a net can have for its cost to be considered during placement. Nets with more than the specified number of pins are not considered.

`?mixedMode { t | nil }`

Allows floating components to be moved when using the *Assisted Mixed CMOS/Standard Cell* placement mode

`?placeCluster` *`t_clusterName`*

Places only the components assigned to the specified cluster name.

`?componentEdge { Boundary | Bounding Box }`

Specifies which edge is used when placing a component against the placement boundary.

`Boundary` places the component's boundary against the placement boundary. This does not include any well spacing defined around the component, which consequently might lie outside the boundary after placement.

`Bounding Box` places the component's bounding box against the placement boundary. This includes any well spacing defined around the component.

Value Returned

`t`	The custom placer was run. Progress and problems are reported using the usual error and warning messages.

Examples

Runs the custom digital placer by using the settings specified by the related environment variables.

```
vcpfeRunCustomDigitalPlacer(schId layId)
```

6

Space-based Router Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso® Space-based Automatic Router.

Only the functions listed below are supported for public use. Any other function, regardless of its name or prefix, and undocumented aspects of the functions described below, are private and subject to change at any time.

The SKILL functions that let you manage the tasks in the Virtuoso® Space-based Automatic Router can be broadly classified in the following categories.

- Antenna Functions - These functions let you check and fix process antenna violations.
  - rteCheckAntenna
  - rteFixAntenna
- Automatic Routing Functions - These functions are used for routing signal nets.
  - rteFixViolations
  - rteFixViolations
  - rteSearchAndRepair
  - rteOptimizeRoute
- Batch Checking and Fixing Functions - These functions let you increase manufacturing reliability and yields by fixing violations and optimizing routing.
  - rteFixViolations
  - rteOptimizeRoute
- Compose/Decompose Trunk Functions - These functions let you convert shapes of selected nets into trunk objects and vice-versa.
- Cover Obstruction Functions - These functions let you turn on and turn off the cover obstruction highlight.

- rtePowerRouteBlockRing
- rteCoverObstructionUnHilite

- Geometry Functions - These functions let you generate and modify shapes.
  - rtePowerRouteBlockRing
  - rteGeomAndNot
  - rtePowerRouteBlockRing
  - rteGeomOr
  - rteGeomSize
  - rteGeomXor
- Power Routing Functions - These functions let you add power components in the design.
  - rtePowerRouteBlockRing
  - rtePowerRouteCellRow
  - rtePowerRouteCoreRing
  - rtePowerRoutePadRing
  - rtePowerRoutePinToTrunk
  - rtePowerRouteStripes
  - rtePowerRouteTrimStripes
  - rtePowerRouteViaInsertion
  - rtePowerRouteTieShield
- Power Routing Scheme Creation Functions - These functions let you manipulate schemes of a power component.
  - rteCreateBlockRingScheme
  - rteCreateCellRowsScheme
  - vsrLoadPreset
  - rteCreatePadRingScheme
  - vsrLoadPreset
  - rteCreateStripesScheme

- vsrLoadPreset

- VSR Preset Functions - These functions let you save, load, and delete routing-related settings and user override constraint values for all Automatic routing features.
  - vsrLoadPreset
  - vsrSavePreset
  - vsrDeletePreset
- Virtuoso Routing Integrated Development Environment Functions - These functions let you work with Routing Integrated Development Environment.
  - rdeCloseSession
  - rdeDone
  - rdeEval
  - rdeGetVar
  - rdeHide
  - rdeInspect
  - rdeOpenCurrentDesign
  - rdeReplay
  - rdeSequencer
  - rdeSetVar
  - rdeShow
  - rdeSource
  - rdeCreateChamferFill
  - rdeCreateWireChamfer
  - Space-based Router Functions

rdeCloseSession

```
rdeCloseSession(
     ) ;
     => t / nil
```

Description

This function terminates the current Virtuoso Routing IDE session and closes the Virtuoso Routing IDE window. The `rdeCloseSession` function returns the GXL tokens if it was checked out by the `rdeShow` function.

Arguments

None

Value Returned

`t`	Returns `t` when the Virtuoso Routing IDE session terminates and the window is closed.
`nil`	Returns `nil` if there was no active Virtuoso Routing IDE session at the time the function was executed.

Examples

Terminates and closes the current Virtuoso Routing IDE session.

```
rdeCloseSession( )
```

Related Topics

Space-based Router Functions

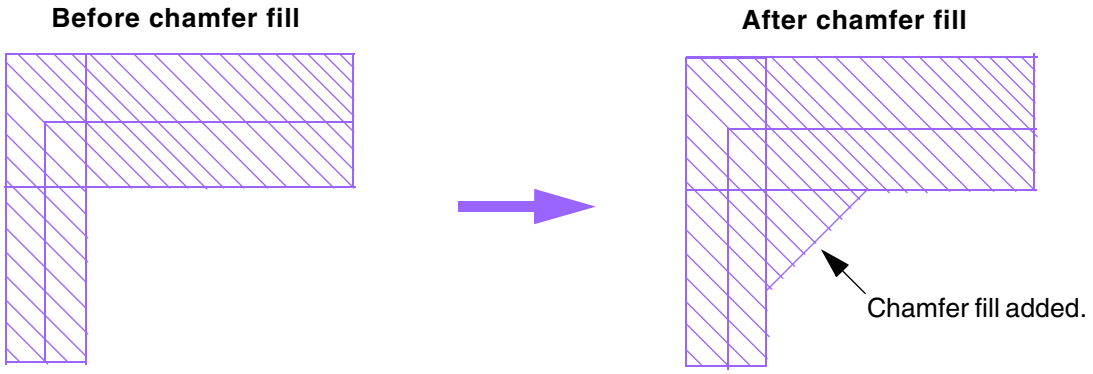
rdeCreateChamferFill

```
rdeCreateChamferFill(
     [ ?all { t | nil } ]
     [ ?cv d_cvid ]
     [ ?region l_region ]
     [ ?layer lt_layer ]
     [ ?lengthThreshold f_length ]
     [ ?chamferValue1 {f_value1 | l_chamfer1} ]
     [ ?chamferValue2 {f_value2 | l_chamfer2} ]
     [ ?extendViaChamfer { t | nil } ]
     [ ?allowViolation { t | nil } ]
     [ ?outputPurpose t_purposeName ]
     [ ?materialRemovalPurpose t_purposeName ]
     [ ?chamferFilter l_filterCriteria ]
     [ ?useEdgeLength { t | nil } ]
     )
     => t / nil
```

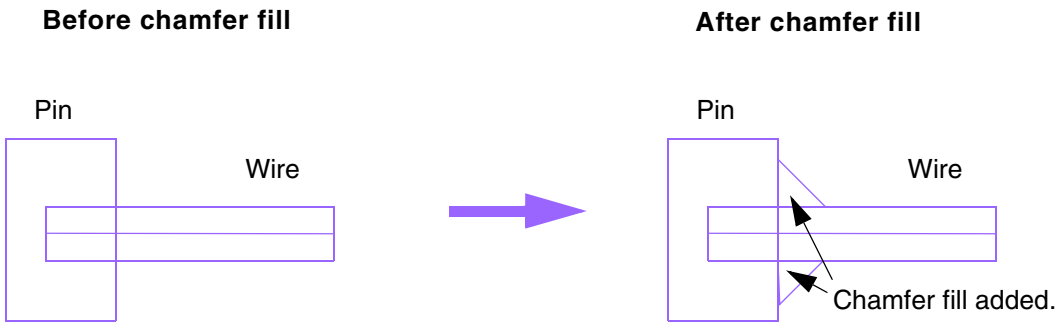
Description

Creates chamfer fill on wires (pathSegs). The chamfer fill mechanism removes 90-degree corners by filling the corners of wires that form T-junctions, L-shaped wires, wire-to-via, wire-to-rectangle, and wire-to-polygon connections. T-junctions and L shapes created using only rectangles or polygons are not considered for chamfer fill.

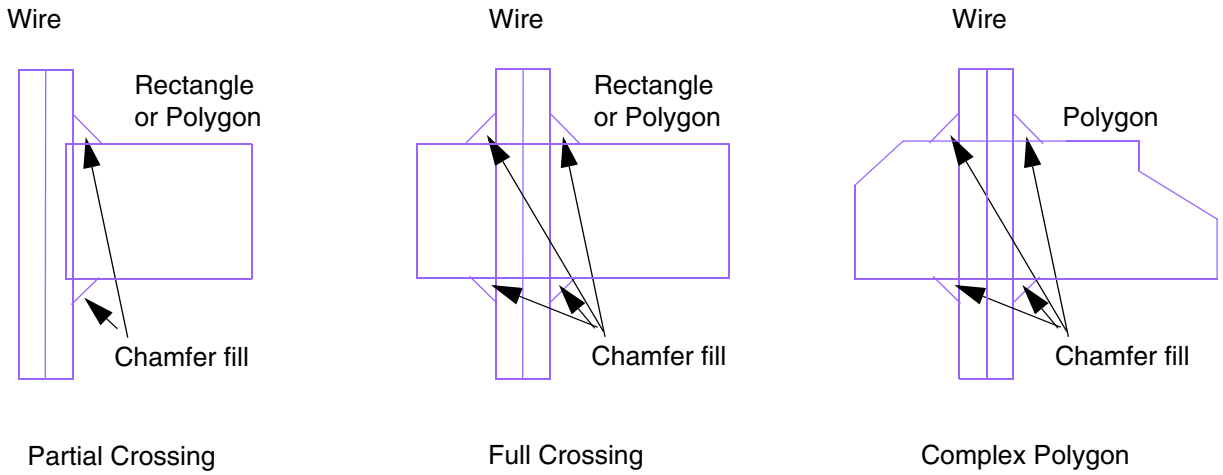
An example of a chamfer fill shape on an L-shaped wire is shown in the following figure.



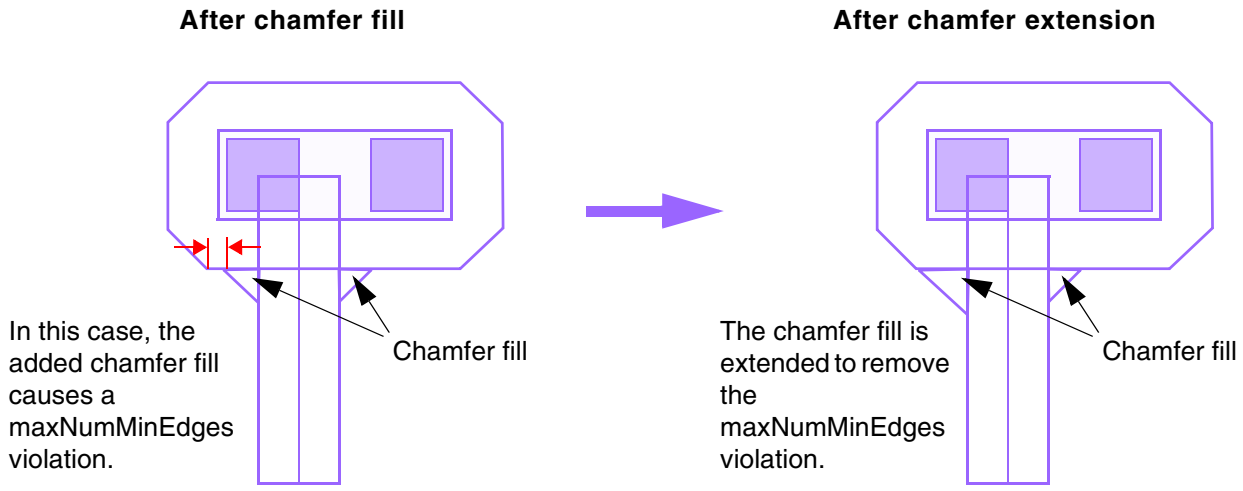
An example of chamfer fill for a wire that connects to a pin (rectangle) is shown in the following figure.



The following figure shows examples of chamfer fill shapes added at the intersections of wires and rectangles or polygons.



If the added chamfer fill at a via junction causes a maxNumMinEdges violation, then the chamfer fill edges are extended to fix the violation, as shown in the following figure.



Arguments

`?all { t | nil }`	If set to `t`, creates chamfer fill on all the wires in the design. **Note:** Specify either `?all` or `?region` but not both . `?all` is the default setting if `?region` is not specified.
`?cv` *`d_cvId`*	Database ID of the cellview. Default is the current active window.
`?region` *`l_region`*	Creates chamfer fill on the wires in the specified region. Specify the region using a list containing a pair of XY coordinates defining the lower-left and upper-right corners of a rectangle in the following format: `list(`*`f_xlo f_ylo f_xhi f_yhi`*`)` **Note:** Specify either `?all` or `?region` but not both. `?all` is the default setting if `?region` is not specified.
`?layer` *`lt_layer`*	Creates chamfer fill on only the layers in the list. By default, chamfer fill is created on all the routing layers.
`?lengthThreshold` *`f_length`*	

Specifies the threshold that determines whether `chamferValue1` or `chamferValue2` is applied.

If `length1` or `length2` is less than the `lengthThreshold` value, `chamferValue1` is applied; otherwise `chamferValue2` is applied. See Example 2—Chamfer fill where lengthThreshold is considered for an example of how `length1` and `length2` are measured.

`?chamferValue1 { f_value1 | l_chamfer1 }`

Specifies the chamfer value to be applied if `length1` or `length2` is less than the `lengthThreshold` value. See Example 2—Chamfer fill where lengthThreshold is considered for an example.

- If a single value is specified, it applies to both the inner and outer chamfers.
- If two values are specified in a list, the first value is the inner chamfer value and the second value is the outer chamfer value.

`?chamferValue2 { f_value2 | l_chamfer2 }`

Specifies the chamfer value to be applied if `length1` and `length2` are greater than or equal to the `lengthThreshold` value. See Example 2—Chamfer fill where lengthThreshold is considered for an example.

- If a single value is specified, it applies to both the inner and outer chamfers,
- If two values are specified in a list, the first value is the inner chamfer value and the second value is the outer chamfer value.

`?extendViaChamfer { t | nil }`

Adds chamfer fill on wire-to-via junctions. The default is `t`.

`?allowViolation { t | nil }`

Adds chamfer fill even if it causes design rule violations. The default is `nil`.

`?outputPurpose t_purposeName`

Specifies the layer purpose on which added chamfer fill shapes are drawn. If `outputPurpose` is not specified, the default purpose `gapFill` is used.

`?materialRemovalPurpose` *`t_purposeName`*

Specifies the layer purpose on which wire removal shapes are drawn when chamfering the external corners of L-shaped wires.

The removal shape is drawn on the specified purpose on the same layer as the wire and is processed when exporting GDSII or during fracturing.

See Example 3—Chamfering external corners of L-shaped wires for more information.

`?chamferFilter` *`l_filterCriteria`*

Filters the wires on which chamfer fill is created, based on a list of specified criteria. For example:

- To create chamfer fill only on wires with maximum voltage greater than 100V, specify
  `?chamferFilter list("voltage > 100.0")`
- To create chamfer fill only on wires with width greater than or equal to 10 microns, specify
  `?chamferFilter list("width >= 10.0")`
- To create chamfer fill only on wires with width greater than 8 microns, and voltage less than 45V, specify
  `?chamferFilter list("width > 8.0" "voltage < 45.0")`

`?useEdgeLength { t | nil}`

Computes the length on the external edges of the wire. The default is `nil`, in which case the centerline of the wire is used.

Value Returned

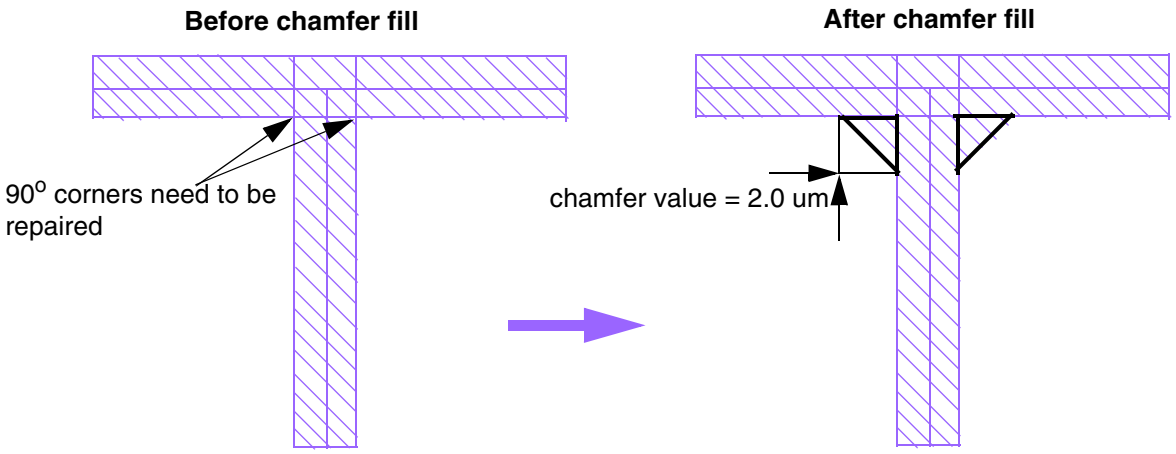
`t`	The specified chamfer fill was created.
`nil`	Chamfer fill creation failed.

Examples

Example 1—Chamfer fill where lengthThreshold is not considered

```
rdeCreateChamferFill(?all t ?lengthThreshold 0.0 ?chamferValue1 2.0 ?chamferValue2
2.0)
```

Creates chamfer fill of 2 microns on all the 90-degree corners of all the wires on all the routing layers. In this scenario, the `lengthThreshold` value (0 microns) is not considered. The following figure illustrates this example with a T-junction wire; chamfer fill of 2 microns is created on both the sides of the T-junction.



Example 2—Chamfer fill where lengthThreshold is considered

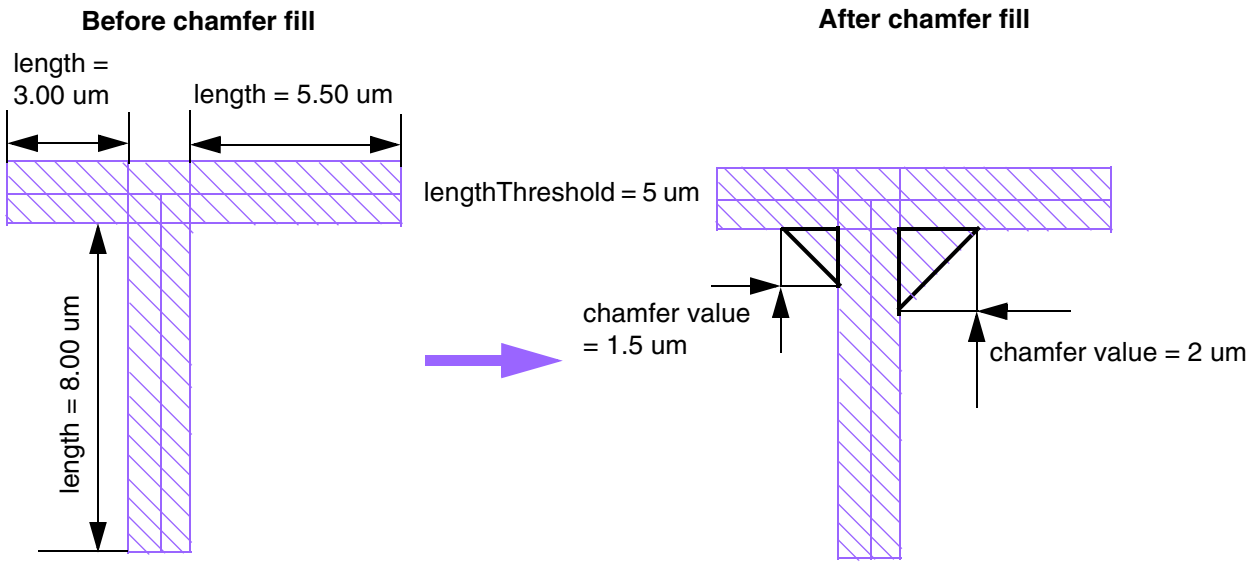
```
rdeCreateChamferFill(?all t ?layer '("M2"} ?lengthThreshold 5.0 ?chamferValue1 1.5
?chamferValue2 2.0)
```

Creates chamfer fill on all the 90-degree corners of all the wires on the `M2` layer according to the following conditions:

- If the length of one segment is less than the `lengthThreshold` value (5 microns), the chamfer fill value will be 1.5 microns.
- If the length of both the segments is greater than or equal to the `lengthThreshold` value (5 microns), the chamfer fill value will be 2 microns.

The following figure illustrates this example with a T-junction wire. The lengths of the segments for the left corner are 3 and 8 microns, so its chamfer fill value is 1.5 microns. The

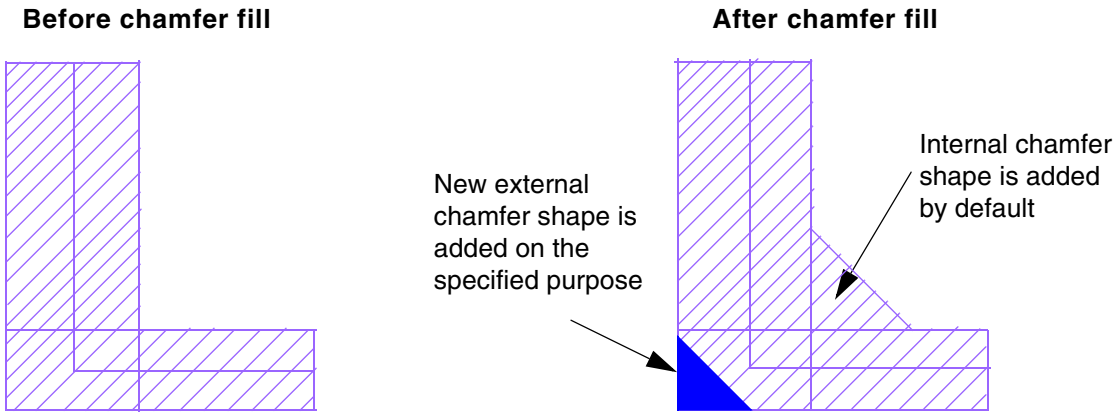
lengths of the segments for the right corner are 5.5 and 8 microns, so its chamfer fill value is 2 microns.



Example 3—Chamfering external corners of L-shaped wires

```
rdeCreateChamferFill(?all t ?layer '("M2"} ?materialRemovalPurpose "noDrawing"
```

The following figure illustrates the effect of specifying the `?materialRemovalPurpose` argument, which specifies the layer purpose on which chamfer removal shapes are drawn on the external corners of wires.



Related Topics

Space-based Router Functions

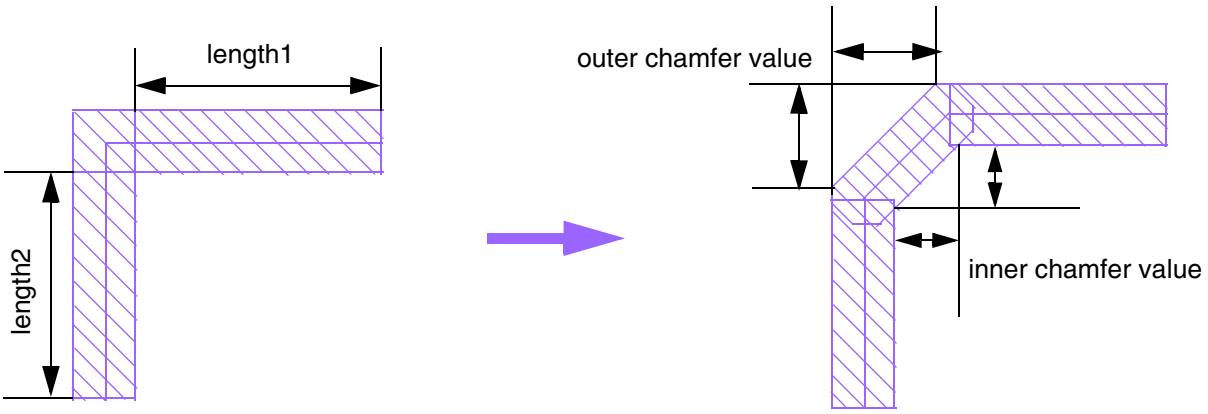
rdeCreateWireChamfer

```
rdeCreateWireChamfer(
     [ ?all { t | nil } ]
     [ ?cv d_cvid ]
     [ ?region l_region ]
     [ ?layer lt_layer ]
     [ ?lengthThreshold f_length ]
     [ ?chamferValue1 {f_value1 | l_chamfer1} ]
     [ ?chamferValue2 {f_value2 | l_chamfer2} ]
     [ ?allowViolation { t | nil } ]
     [ ?innerChamfer { t | nil } ]
     [ ?chamferFilter l_filterCriteria ]
     [ ?chamferEndOfStripe { t | nil } ]
     )
     => t / nil
```

Description

Creates chamfers on wires (pathSegs). The wire chamfer mechanism replaces 90-degree wires with 45-degree wires.

An example of a wire chamfer is shown in the following figure.



Arguments

`?all { t | nil }`	If set to `t`, creates chamfers on all the wires in the design. **Note:** Specify either `?all` or `?region` but not both. `?all` is the default setting if `?region` is not specified.
`?cv` *`d_cvId`*	Database ID of the cellview. Default is the current active window.

`?region` *`l_region`*

Creates chamfers on the wires in the specified region. Specify the region using a list containing a pair of XY coordinates defining the lower-left and upper-right corners of a rectangle in the following format:

`list(`*`f_xlo f_ylo f_xhi f_yhi`*`)`

Specify either `?all` or `?region` but not both. `?all` is the default setting if `?region` is not specified.

`?layer` *`lt_layer`*

Creates chamfers on the layers in the list. By default, chamfers are created on all the routing layers.

`?lengthThreshold` *`f_length`*

Specifies the threshold value that determines if `chamferValue1` or `chamferValue2` is applied.

If `length1` or `length2` is less than the `lengthThreshold` value, `chamferValue1` is applied; otherwise `chamferValue2` is applied.

`?chamferValue1 {` *`f_value1`* `|` *`l_chamfer1`* `}`

Specifies the chamfer value to be applied if `length1` or `length2` is less than the `lengthThreshold` value. See Example 2—Chamfer fill where lengthThreshold is considered for an example.

- If a single value is specified, it is the inner, outer, and end-of-stripe chamfer.
- If two values are specified in a list, the first value is the inner and outer chamfer, and the second value is the end-of-stripe value.

`?chamferValue2 {` *`f_value2`* `|` *`l_chamfer2`* `}`

Specifies the chamfer value to be applied if `length1` and `length2` are greater than or equal to the `lengthThreshold` value. See Example 1—Chamfer fill where lengthThreshold is not considered for an example.

- If a single value is specified, it is the inner, outer, and end-of-stripe chamfer.
- If two values are specified in a list, the first value is the inner and outer chamfer, and the second value is the end-of-stripe value.

`?allowViolation { t | nil }`

Adds chamfers even if it causes design rule violations. The default is `nil`.

`?innerChamfer { t | nil }`

If set to `nil`, `chamferValue1` or `chamferValue2` applies to the outer corner of the wire. The default value `t` applies `chamferValue1` or `chamferValue2` to the inner corner of the wire.

`?chamferFilter` *`l_filterCriteria`*

Filters the wires on which chamfers are created based on specified criteria. For example:

- To create chamfers only on wires with maximum voltage greater than 100V, specify
  `?chamferFilter list("voltage > 100.0")`
- To create chamfers only on wires with width greater than or equal to 10 microns, specify
  `?chamferFilter list("width >= 10.0")`
- To create chamfers only on wires with width greater than 8 microns, and voltage less than 45V, specify
  `?chamferFilter list("width > 8.0" "voltage < 45.0")`

`?chamferEndOfStripe { t | nil }`

When set to `t`, ends of wires are chamfered to match chamfered via arrays. By default (`nil`), ends of wires are not chamfered.

See Example 3—Chamfering end of stripe wires for more information.

Value Returned

`t`	Wire chamfers were created.
`nil`	Wire chamfer creation failed.

Examples

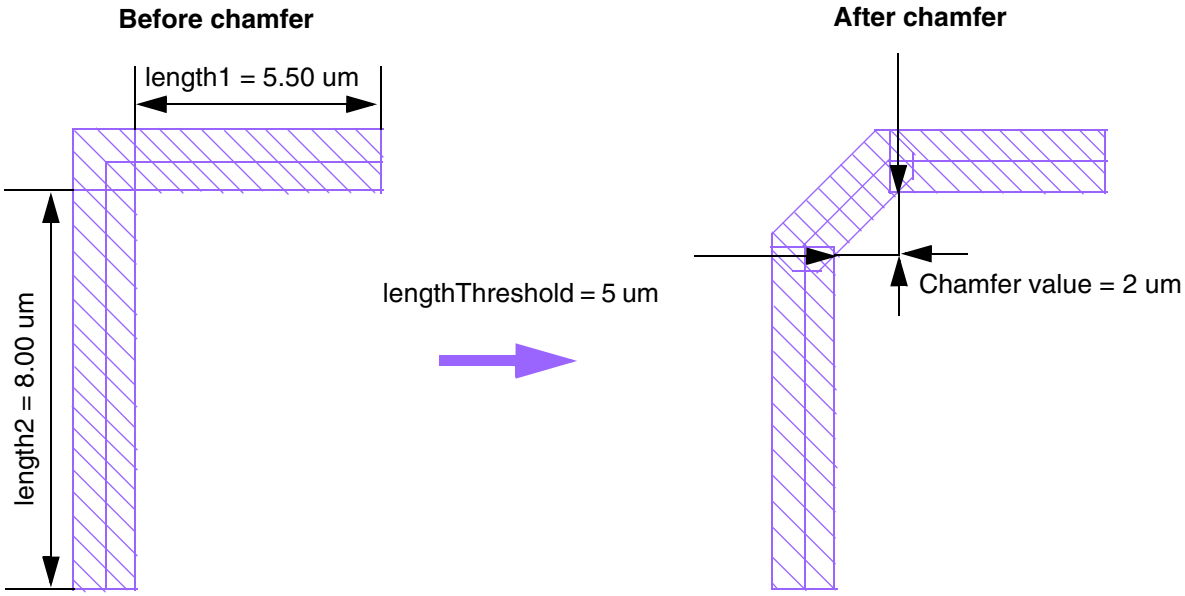
Example 1—Inner wire chamfer

```
rdeCreateWireChamfer(?all t ?layer '("M2") ?lengthThreshold 5.0 ?chamferValue1 1.5
?chamferValue2 2.0 ?innerChamfer t)
```

Creates chamfers on the inner corners of all the wires on the `M2` layer according to the following conditions:

- If the length of one segment is less than the `lengthThreshold` value (5 microns), the chamfer value will be 1.5 microns.
- If the length of both the segments is greater than or equal to the `lengthThreshold` value (5 microns), the chamfer value will be 2 microns.

In the following figure, the length of both the segments (5.5 and 8 microns) is greater than the `lengthThreshold` value (5 microns), so the inner chamfer value is 2 microns.



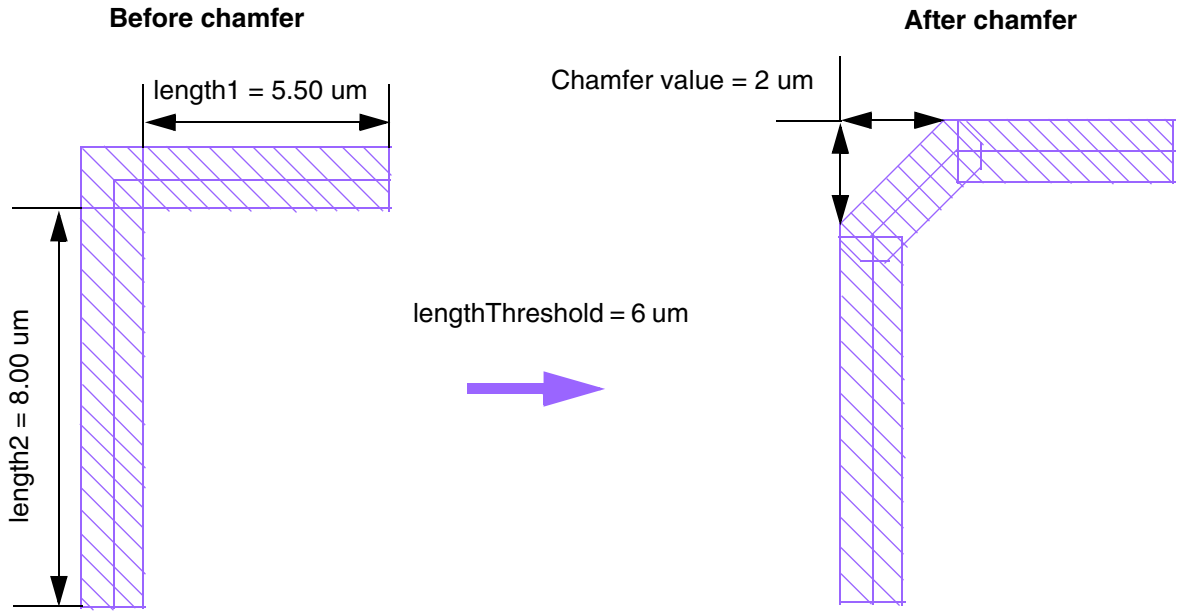
Example 2—Outer wire chamfer Where length1 Is Less Than lengthThreshold

```
rdeCreateWireChamfer(?all t ?layer list("M2") ?lengthThreshold 6.0 ?chamferValue1
2.0 ?chamferValue2 3.0 ?innerChamfer nil)
```

Creates wire chamfers on the outer side of all the wires on the `M2` layer according to the following conditions:

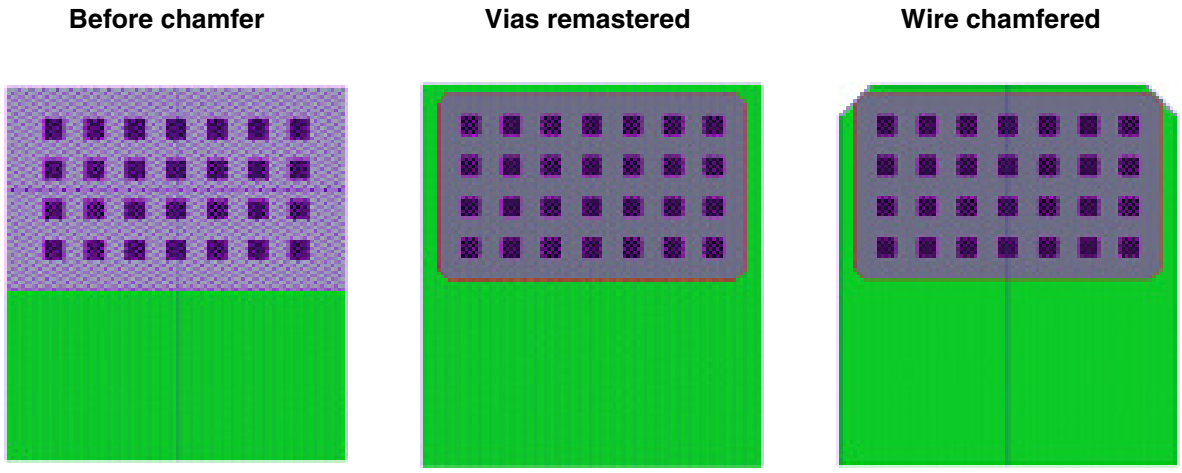
- If the length of one segment is less than the `lengthThreshold` value (6 microns), the chamfer value will be 2 microns.
- If the length of both the segments is greater than or equal to the `lengthThreshold` value (6 microns), the chamfer value will be 3 microns.

In the following figure, the length of one segment (5.5 microns) is less than the `lengthThreshold` value (6 microns), so the outer chamfer value is 2 microns.



Example 3—Chamfering end of stripe wires

The following figure illustrates the effect of specifying the `chamferEndOfStripe` argument, which chamfers the ends of stripes to match chamfered via arrays.



Related Topics

Space-based Router Functions

rdeDone

```
rdeDone(
     ) ;
     => t / nil
```

Description

This function closes the Virtuoso Routing IDE system.

Arguments

None

Value Returned

`t`	Returns `t` if Virtuoso Routing IDE was active and was closed properly.
`nil`	Returns `nil` if Virtuoso Routing IDE was inactive at the time the function was executed.

Example

Closes the Virtuoso Routing IDE application.

```
rdeDone( )
```

Related Topics

Space-based Router Functions

rdeEval

```
rdeEval(
     t_expression
     [ g_openCurrentDesign ]
     [ s_doCheckPoint ]
     ) ;
     => t / nil
```

Description

This function evaluates the Tcl expression that is supplied. There are no arguments to open the current design and also perform a checkpoint once the Tcl expression is evaluated.

Arguments

t_expression	Evaluates the Tcl expression that is supplied.
g_openCurrentDesign	Loads the design that corresponds to the current Graphics Editor window into Virtuoso Routing IDE. This argument can be set to `t` or `nil`. By default, it is set to `t`.
s_doCheckPoint	Performs an automatic checkpoint of the design data.This argument can be set to `t` or `nil`. By default, it is set to `t`.

Value Returned

`t`	Returns `t` if Virtuoso Routing IDE is started
`nil`	Returns `nil` if it was unable to start Virtuoso Routing IDE.

Examples

Loads the design that corresponds to current GE window into Virtuoso Routing IDE, runs the Tcl command to perform bus routing and does an automatic checkpoint.

```
rdeEval("bus_route")
```

Loads the design that corresponds to current GE window into Virtuoso Routing IDE, runs the Tcl command to perform bus routing and does an automatic checkpoint.

```
rdeEval("bus_route" t t)
```

Assuming that Virtuoso Routing IDE is already open, run the Tcl command to perform bus routing but does not perform an automatic checkpoint.

```
rdeEval("bus_route" nil nil)
```

Related Topics

Space-based Router Functions

rdeGetVar

```
rdeGetVar(
     [ t_varname ]
     ) ;
     => t / nil
```

Description

This function returns the Virtuoso Routing IDE environment variable. The Virtuoso Routing IDE environment variables are independent of the dfII environment package.

Arguments

t_varname	The name of the environment variable.

Value Returned

`t`	Returns `t` if the value of the environment variable could be found.
`nil`	Returns `nil` if it was unable to find the value of the environment variable.

Examples

The following example looks for the value of the `db.user_namespace` environment variable.

```
rdeGetVar "db.user_namespace"
```

Related Topics

Space-based Router Functions

rdeHide

```
rdeHide(
     ) ;
     => t / nil
```

Description

This function terminates the current Virtuoso Routing IDE session and closes the Virtuoso Routing IDE window. The `rdeHide` function returns the GXL tokens if it was checked out by the `rdeShow` function.

Arguments

None

Value Returned

`t`	Returns `t` when the Virtuoso Routing IDE session terminates and the window is closed.
`nil`	Returns `nil` if there was no active Virtuoso Routing IDE session at the time the function was executed.

Examples

Terminates and hides the current Virtuoso Routing IDE session.

```
rdeHide( )
```

Related Topics

Space-based Router Functions

rdeInspect

```
rdeInspect(
     [ d_dbID ]
     ) ;
     => t / nil
```

Description

This function displays the Virtuoso Routing IDE property inspector GUI. If a dbObject is supplied, then the inspector will show that object and its properties. If no object is supplied, then the currently open geWindow's cellview will be inspected.

Note: The rdeInspect() SKILL function only displays the property inspector on a single object and not on a set of objects.

Arguments

d_dbID	The ID of the database object that needs to be inspected.

Value Returned

`t`	Returns `t` if the dbObject of the specified ID could be found.
`nil`	Returns `nil` if If there is no current geWindow or no supplied dbObject.

Examples

To invoke the inspector on an object and then click the object in the Layout Editor.

```
rdeInspect (gePointToFig())
```

Related Topics

Space-based Router Functions

rdeOpenCurrentDesign

```
rdeOpenCurrentDesign(
     [ g_showWindow ]
     ) ;
     => t / nil
```

Description

This function will open the Virtuoso Routing IDE window and start the Virtuoso Routing IDE environment on the cellview that the geWindow has currently loaded. It will also make the current Virtuoso window `ReadOnly` until the window is closed.

Arguments

g_showWindow	Allows you to display the Virtuoso Routing IDE GUI. If this argument is set to `nil`, the GUI will not be shown.

Value Returned

`t`	Returns `t` if the Virtuoso Routing IDE GUI is opened.
`nil`	Returns `nil` if the GUI is not displayed.

Examples

Loads the design that the geWindow has currently loaded and starts the Virtuoso Routing IDE GUI.

```
rdeOpenCurrentDesign()
```

Loads the design that the geWindow has currently loaded but does not display the Virtuoso Routing IDE GUI.

```
rdeOpenCurrentDesign(nil)
```

Related Topics

Space-based Router Functions

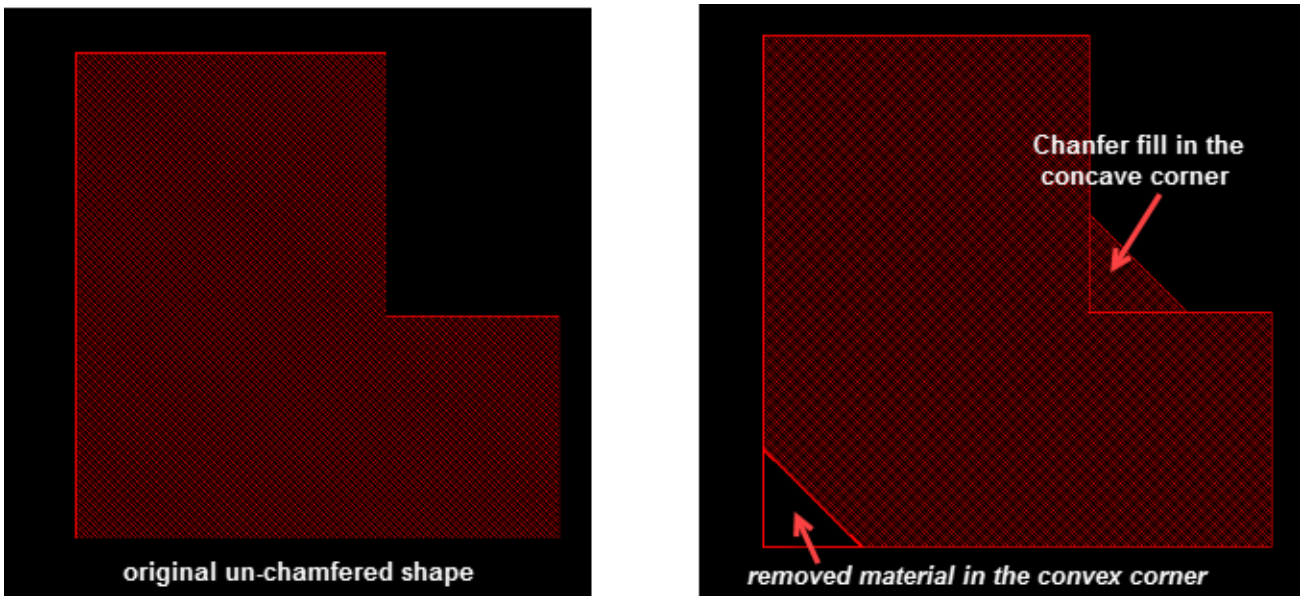
rdeRemoveChamferFill

```
rdeRemoveChamferFill(
     [ ?all { t | nil } ]
     [ ?region l_region ]
     [ ?layer lt_layer ]
     [ ?outputPurpose t_purposeName ]
     [ ?materialRemovalPurpose t_purposeName ]
     [ ?concaveOnly { t | nil }
     )
     => t / nil
```

Description

Removes all the right-angled triangle shapes of purposes specified by `outputPurpose` and `materialRemovalPurpose` (except if `concaveOnly` is true). Usually, these shapes are created by the command rdeCreateChamferFill.

The following figure shows an example of chamfer removal.



After the `rdeRemoveChamferFill` command, the original shape is attained.

Arguments

`?all { t | nil }`	If set to `t`, removes all chamfer fill and remove material shapes whatever the location is in a concave or convex corner shapes. **Note:** Specify either `?all` or `?region` but not both . `?all` is the default setting if `?region` is not specified.
`?region` *`l_region`*	Removes all chamfer fill and remove material shapes whatever the location is in a concave or convex corner shapes. Specify the region using a list containing a pair of XY coordinates defining the lower-left and upper-right corners of a rectangle in the following format: `list(`*`f_xlo f_ylo f_xhi f_yhi`*`)` **Note:** Specify either `?all` or `?region` but not both. `?all` is the default setting if `?region` is not specified.
`?layer` *`lt_layer`*	Removes chamfer fill on only the layers in the list. By default, chamfer fill is removed on all the routing layers.
`?outputPurpose` *`t_purposeName`*	Specifies the layer purpose from which chamfer fill shapes are removed. If `outputPurpose` is not specified, the default purpose `drawing` is used.
`?materialRemovalPurpose` *`t_purposeName`*	Specifies the layer purpose of the shapes being used to chamfer convex corners. If specified, the shapes are removed.
`?concaveOnly { t | nil }`	Removes the concave and convex shapes. Default is `nil`. If `t` is specified, only concave shapes are removed and not the convex ones.

Value Returned

`t`	The specified chamfer fill was removed.

`nil`	Chamfer fill removal failed.

Examples

The following example removes the right-angled triangle shapes of purpose `fill`.

```
rdeRemoveChamferFill(?all ?outputPurpose "fill" ?concaveOnly t)
```

Related Topics

Space-based Router Functions

rdeReplay

```
rdeReplay(
     t_fileName
     [ g_openCurrentDesign ]
     [ s_doCheckPoint ]
     ) ;
     => t / nil
```

Description

This function reads the supplied file name and send each line one-at-a-time to the Tcl interpreter. This is in contrast to rdeSource, in which the entire Tcl file is read into the interpreter and then executed. It may be useful in determining which line of a script file had an error, in case an error occurs. Since each line is executed independently, it is not possible to create expressions (such as function definitions) that span more than one line.

Arguments

t_fileName	The name of the file to be executed.
s_openCurrentDesign	Allows you to load the current geWindow design in the Virtuoso Routing IDE environment. This argument can be set to `t` or `nil`. By default, the option is set to `t`. If this option is set to `nil`, then the current geWindow design will not be loaded.
s_doCheckPoint	Performs an automatic checkpoint of the design data unless this argument is supplied and set to `'nil'`. This argument can be set to `t` or `nil`. By default, it is set to `t`.

Value Returned

`t`	Returns `t` if the file is replayed.
`nil`	Returns `nil` if the file is not replayed.

Examples

The following example opens the current design, executes the command in `test.tcl` and does an automatic checkpoint.

```
rdeReplay("test.tcl")
```

Related Topics

Space-based Router Functions

rdeSequencer

```
rdeSequencer(
     [ t_designStyle ]
     ) ;
     => t / nil
```

Description

This function will display the auto router sequencer dialog box. It will obtain some values from the current geWindow. If there is no current geWindow, it will return nil.

The sequencer does not acquire any license until the sequence is executed completely.

Arguments

t_designStyle	The design style of the sequence presets can be supplied by the optional argument. The option argument can be one of "default", "custom", "asic", "device_level", or "chip_assembly". If the design style is not supplied, the style will be determined semi-automatically.

Value Returned

`t`	Returns `t` if the auto router sequencer dialog box is displayed.
`nil`	Returns `nil` if the auto router sequencer dialog box cannot be displayed.

Examples

Displays the auto router sequencer dialog box.

```
rdeSequencer("asic")
```

Related Topics

Space-based Router Functions

rdeSetVar

```
rdeSetVar(
     t_varName
     g_varValue
     ) ;
     => t / nil
```

Description

This function sets the Virtuoso Routing IDE environment variable to the supplied value. The Virtuoso Routing IDE environment variables are independent of the dfII environment package.

Arguments

t_varName	The name of the environment variable that is to be set.
g_varValue	The value of the environment variable that is to be set.

Value Returned

`t`	Returns `t` if the environment variable could be set.
`nil`	Returns `nil` if it is unable to set the environment variable.

Examples

Sets the specified Virtuoso Routing IDE environment variable to `"CDBA"`.

```
rdeSetVar "db.user_namespace" "CDBA"
```

Related Topics

Space-based Router Functions

rdeShow

```
rdeShow(
     ) ;
     => t / nil
```

Description

This function displays the Virtuoso Routing IDE window.

Arguments

None

Value Returned

`t`	Returns `t` if the Virtuoso Routing IDE window is displayed.
`nil`	Returns `nil` if the Virtuoso Routing IDE window does not appear.

Examples

Displays the Virtuoso Routing IDE window.

```
rdeShow( )
```

Related Topics

Space-based Router Functions

rdeSource

```
rdeSource(
     t_fileName
     [ g_openCurrentDesign ]
     [ s_doCheckPoint ]
     ) ;
     => t / nil
```

Description

This function sources the Tcl expression that is supplied. There are arguments to open the current design and also perform a checkpoint once the Tcl file has been sourced.

Arguments

t_fileName	The name of the file to be executed.
g_openCurrentDesign	Allows you to load the current geWindow design in Virtuoso Routing IDE environment. This argument can be set to `t` or `nil`. By default, the option is set to `t`. If this option is set to `nil`, then the current geWindow design will not be loaded.
s_doCheckPoint	Performs an automatic checkpoint of the design data unless this argument is supplied and set to `'nil'`. This argument can be set to `t` or `nil`. By default, it is set to `t`.

Value Returned

`t`	Returns `t` if the Virtuoso Routing IDE environment is initiated.
`nil`	Returns `nil` if the supplied Tcl script file is not found.

Examples

The following example opens the current design, sources the Tcl file `test.tcl` and does an automatic check.

```
rdeSource("test.tcl" t t)
```

Related Topics

Space-based Router Functions

rteCheckAntenna

```
rteCheckAntenna(
     [ ?cv d_cvid ]
     [ ?all g_all ]
     [ ?nets l_nets ]
     [ ?set g_set ]
     [ ?excludeNets l_excludeNets ]
     [ ?model l_models ]
     ) ;
     => t / nil
```

Description

Checks for process antenna violations for the entire design, specific nets, or nets in a set. You can choose to ignore specific nets.

To use this command, your design data must have been imported from LEF and DEF files that include process antenna keywords for setting values used by this function.

Arguments

?cv *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Default is the current cellview.
?all *g_all*	Performs process antenna checks on all nets. Values are `t` or `nil` (default). If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.
?nets *l_nets*	Performs process antenna checks on the specified list of nets. If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.
?set *g_set*	Performs process antenna checks on nets in the selected set. Use the Navigator to select the nets. Values are `t` or `nil` (default). If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.
?excludeNets *l_nets*	Excludes the specified list of nets during the check.

`?model` *`l_models`*	Specifies the model(s) to use: OXIDE1, OXIDE2, OXIDE3, OXIDE4. Default is OXIDE1.

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully or an error occurred.

Examples

```
rteCheckAntenna(
?cv (geGetRoutedCellView)
?all t
?excludeNets list("net20" "net16")
?model list("OXIDE1" "OXIDE3")
)
```

Related Topics

Space-based Router Functions

rteCheckDataForRouting

```
rteCheckDataForRouting(
     [ ?mode t_mode ]
     [ ?designRuleSpec t_designRuleSpec ]
     [ ?fromRoutingLayer t_fromRoutingLayer ]
     [ ?toRoutingLayer t_toRoutingLayer ]
     [ ?annotate g_annotate ]
     [ ?reportFileName t_reportFileName ]
     [ ?selected g_selected ]
     ) ;
     => t / nil
```

Description

Checks the technology data and the design data to ensure that the VSR router is able to perform correctly.

Arguments

`?mode` *t_mode*
: Switches between technology checks such as layer information or design checks such as pin spacing. The possible values are `tech` or `design`. The default value is `tech`.

`?designRuleSpec` *t_designRuleSpec*
: Specifies the design rule specification to be used to get constraints information.

`?fromRoutingLayer` *t_fromRoutingLayer*
: Specifies the routing layer from which the check will start. If this option is not specified, the routing layer is taken from the valid routing layers list.

`?toRoutingLayer` *t_toRoutingLayer*
: Specifies the routing layer at which the check will stop. If this option is not specified, the routing layer is taken from the valid routing layers list.

`?annotate` *g_annotate*	Specifies if the checker should output OA markers for some of the design checks. Values are `t` or `nil`. The default value is `t`.
`?reportFileName` *t_reportFileName*	Specifies the XML report file output.
`?selected` *g_selected*	Specifies if the checker should only use the current selected objects.

Value Returned

`t`	No errors have been found while checking the data.
`nil`	Errors were found while checking the data.

Examples

```
rteCheckDataForRouting ?mode "design" ?annotate t
```

```
rteCheckDataForRouting ?mode "tech" ?designRuleSpec "virtuosoDefaultSetup"
?fromRoutingLayer "M2" ?toRoutingLayer "M4" ?reportFileName "techDataCheck.xml"
```

Related Topics

Space-based Router Functions

rteCheckRoutability

```
rteCheckRoutability(
     [ ?cv d_cvId ]
     [ ?noBlockageCheck g_noBlockageCheck ]
     [ ?noMinSpaceCheck g_noMinSpaceCheck ]
     [ ?noMinWidthCheck g_noMinWidthCheck ]
     [ ?noViaCheck g_noViaCheck ]
     ) ;
     => t / nil
```

Description

Checks the design for known conditions that can cause potential routing problems. Design Rule Checks applied to pins and some routability checks are performed primarily to determine the accessibility of pins. You can exclude any of the checks by specifying the appropriate argument. Annotations are automatically created for violations.

Arguments

`?cv` *`d_cvId`*
: Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Default is the current cellview.

`?noBlockageCheck` *`g_noBlockageCheck`*
: Disable pin blockage checking. Values are `t` or `nil` (default).

`?noMinSpaceCheck` *`g_noMinSpaceCheck`*
: Disables minimum pin spacing checking. Values are `t` or `nil` (default).

`?noMinWidthCheck` *`g_noMinWidthCheck`*
: Disables minimum pin width checking. Values are `t` or `nil` (default).

`?noViaCheck` *`g_noViaCheck`*
: Disables via escape checking. Values are `t` or `nil` (default).

Additional Information

A Routability Report that summarizes the issues found is output to the CIW. For example,

```
+-----------------------------------------+
| Routability Report Summary              |
+-----------------------------------------+
|       3 Min spacing violation(s)        |
|       2 Min width violation(s)          |
|       2 Via violations(s)               |
|       0 Blockage violation(s)           |
+-----------------------------------------+
|    9831 pin(s) found                    |
|    9066 pin(s) checked                  |
|     746 pin(s) already connected (skipped) |
|      19 pins(s) not used (skipped)      |
+-----------------------------------------+
```

Use the Annotation Browser to view the violations. Issues such as DRC violations should be dealt with before attempting to route the design.

You can limit the number of markers for each violation type by setting the checkRoutabilityMarkersLimit environment variable. However, the function still produces the number of violations shown in the Routability Report.

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Examples

```
rteCheckRoutability(
?cv (geGetRoutedCellView)
?noViaCheck t
)
```

Related Topics

Space-based Router Functions

rteComposeTrunks

```
rteComposeTrunks(
     ) ;
     => t / nil
```

Description

Converts pathSegs and vias to power trunks usable by both the power and signal routers. The function uses the current cellview and the selected set.

Converts only pathSegs and vias that are in routes. Converted pathSegs and vias are removed from their routes and the routes are repaired.

pathSegs having a topology of `stripe`, `ring`, or `blockRing` have their topology preserved during conversion to power trunks. Otherwise, the converted pathSegs have a topology of `stripe`.

The output specifies the following.

- number of pathSegs selected and the number converted
- number of vias selected and the number converted
- total number of objects selected

Arguments

None

Value Returned

`t`	The function completed successfully with no problems encountered. However, it is possible that no pathSegs or vias were converted. Output messages give the conversion statistics.
`nil`	The function encountered problems. Often, a warning message displays.

Examples

Select the objects in the cellview to compose.

```
rteComposeTrunks()
```

Related Topics

Space-based Router Functions

rteCoverObstructionHilite

```
rteCoverObstructionHilite (
     )
     => none
```

Description

Turns on the cover obstruction hilite sets if at least one cover obstruction has been defined in the top-level or masters.

Arguments

None

Value Returned

`t`	The command executes successfully.
`nil`	The command does not executes successfully.

Examples

Turns on the cover obstruction hilite.

```
rteCoverObstructionHilite()
```

Related Topics

Space-based Router Functions

rteCoverObstructionUnHilite

```
rteCoverObstructionUnHilite (
     )
     => none
```

Description

Turns off the cover obstruction hilite sets for the ones that are already turned on, if at least one cover obstruction has been defined in the top-level or masters.

Arguments

None

Value Returned

`t`	The command executes successfully.
`nil`	The command does not executes successfully.

Examples

```
rteCoverObstructionUnHilite()
```

Related Topics

Space-based Router Functions

rteCreateBlockRingScheme

```
rteCreateBlockRingScheme(
     [ ?name s_name ]
     [ ?contour g_contour ]
     [ ?channel g_channel ]
     [ ?blockClearance f_blockClearance ]
     [ ?horiLayer t_horiLayer ]
     [ ?vertLayer t_vertLayer ]
     [ ?latticeStyle g_latticeStyle ]
     [ ?netClearance f_netClearance ]
     [ ?netWidth f_netWidth ]
     ) ;
     => t / nil
```

Description

Creates a block ring routing scheme.

Arguments

`?name` *s_name*	Name of the scheme. Value is a quoted symbol.
`?contour` *g_contour*	Causes power nets to follow the contour of block instances. Values are `t` or `nil`. If no value is specified, uses the value of the *prouteBlockRingContour* environment variable.
`?channel` *g_channel*	Adds rails in the channels between block instances. Values are `t` or `nil`. If no value is specified, uses the value of the *prouteBlockRingChannels* environment variable.
`?blockClearance` *f_blockClearance*	Separate the block rings and prBoundary of the block instance by the clearance value. The bounding box of the block is not considered, clearance is to the prBoundary. Value is a positive, non-zero floating number. If no value is specified, uses the value of the *prouteBlockRingBlockClearance* environment variable.

`?horiLayer` *`t_horiLayer`*	The horizontal metal layer to use to route the core ring. Value is a layer string. If no value is specified, uses the value of the *prouteBlockRingHorizlayer* environment variable.
`?vertLayer` *`t_vertLayer`*	The vertical metal layer to use to route the core ring. Value is a layer string. If no value is specified, uses the value of the *prouteBlockRingVertlayer* environment variable.
`?latticeStyle` *`g_latticeStyle`*	Extend duplicated net segments to or from a lattice. Values are `t` or `nil`. If set to `nil`, the segments are concentric. If no value is specified, uses the value of the *prouteBlockRingLattice* environment variable.
`?netClearance` *`f_netClearance`*	The minimum clearance between any two power nets. Value is a positive, non-zero floating number. If no value is specified, uses the value of the *prouteBlockRingNetClearance* environment variable.
`?netWidth` *`f_netWidth`*	The total width of the nets to be routed. Value is a positive, non-zero floating number. If no value is specified, uses the value of the *prouteBlockRingNetWidth* environment variable.

Value Returned

`t`	The block ring scheme was created successfully.
`nil`	An error occurred and the block ring scheme was not created.

Examples

```
rteCreateBlockRingScheme(
?name 'schemeOne
?contour t
?channel t
?blockClearance 1.000000
?horiLayer "Metal1"
```

```
?vertLayer "Metal_8"
?latticeStyle t
?netClearance 1.000000
?netWidth 1.000000
)
```

Related Topics

Space-based Router Functions

rteCreateCellRowsScheme

```
rteCreateCellRowsScheme(
     [ ?name s_name ]
     [ ?allRouteLayers g_allRouteLayers ]
     [ ?routeLayers l_routeLayers ]
     [ ?endOfRow g_endOfRow ]
     [ ?extendToNearest g_extendToNearest ]
     ) ;
     => t / nil
```

Description

Creates a cell rows routing scheme.

Arguments

`?name` *`s_name`*
: Name of the scheme. Value is a quoted symbol.

`?allRouteLayers` *`g_allRouteLayers`*
: Use all the layers for cell row routing. Values are `t` or `nil`. If `nil`, the router uses the layers specified by the `?routeLayers` argument. If no value is specified, uses the value of the *prouteCellRowAllLayers* environment variable.

`?routeLayers` *`l_routeLayers`*
: The layers to use to route the cell rows. Value is a list of layer strings. Separate layers with spaces.
The `?allRouteLayers` argument must be set to `nil` to use this argument. If no value is specified, uses the value of the *prouteCellRowLayers* environment variable.

`?endOfRow` *`g_endOfRow`*
: Route standard cells to the end of defined rows. Values are `t` or `nil`. If `nil`, standard cells are routed to the last cell in the row. If no value is specified, uses the value of the *prouteCellRowRowEnd* environment variable.

`?extendToNearest` *`g_extendToNearest`*
: Route the cell rows to the power rail. If more than one rail exists, routes to the farthest rail. Values are `t` or `nil`. If no value is specified, uses the value of the *prouteCellRowExtend* environment variable.

Value Returned

`t`	The cell rows scheme was created successfully.
`nil`	An error occurred and the cell rows scheme was not created.

Examples

```
rteCreateCellRowsScheme(
?name 'schemeOne
?allRouteLayers nil
?routeLayers '("Metal3" "Metal4" "Metal5" "Metal6")
?endOfRow t
?extendToNearest nil
)
```

Related Topics

Space-based Router Functions

rteCreateCoreRingScheme

```
rteCreateCoreRingScheme(
     [ ?name s_name ]
     [ ?ringAtCenter g_ringAtCenter ]
     [ ?relativeTo s_relativeTo ]
     [ ?coreClearance f_coreClearance ]
     [ ?padClearance f_padClearance ]
     [ ?inAreaClearance f_inAreaClearance ]
     [ ?outAreaClearance f_outAreaClearance ]
     [ ?areaXLo f_areaXLo ]
     [ ?areaYLo f_areaYLo ]
     [ ?areaXHi f_areaXHi ]
     [ ?areaYHi f_areaYHi ]
     [ ?horiLayer t_horiLayer ]
     [ ?vertLayer t_vertLayer ]
     [ ?latticeStyle g_latticeStyle ]
     [ ?netClearance f_netClearance ]
     [ ?netWidth f_netWidth ]
     ) ;
     => t / nil
```

Description

Creates a core ring routing scheme.

Arguments

`?name` *`s_name`*
: Name of the scheme. Value is a quoted symbol.

`?ringAtCenter` *`g_ringAtCenter`*
: Routes a ring between the core area of the design and the surrounding pad instances. Values are `t` or `nil`. If `nil`, the ring is routed according to the setting of the `?relativeTo` argument.

  If no value is specified, uses the value of the *prouteCoreRingAtCenter* environment variable.

`?relativeTo` *`s_relativeTo`*

The location of the ring is relative to the specified location. The values are the following:

‘insideIOPads
‘outsideCore
‘insideRegion
‘outsideRegion
With insideRegion or outsideRegion, also specify the region using the ?areaX/YLo and ?areaX/YHi arguments. You must also specify a positive, non zero clearance using one of the following four arguments.

If no value is specified, uses the value of the *prouteCoreRingRelativeTo* environment variable.

`?coreClearance` *`f_coreClearance`*

Use when `?relativeTo` is set to `‘outsideCore`. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteCoreRingCoreClearance* environment variable.

`?padClearance` *`f_padClearance`*

Use when `?relativeTo` is set to `‘insideIOPads`. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteCoreRingPadClearance* environment variable.

`?inAreaClearance` *`f_inAreaClearance`*

Use when `?relativeTo` is set to `‘insideRegion`. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteCoreRingInAreaClearance* environment variable.

`?outAreaClearance` *`f_outAreaClearance`*

Use when `?relativeTo` is set to `‘outsideRegion`. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteCoreRingOutAreaClearance* environment variable.

`?areaXLo` *`f_areaXLo`*	Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. If no value is specified, uses the value of the *prouteCoreRingRTXLo* environment variable.
`?areaYLo` *`f_areaYLo`*	Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. If no value is specified, uses the value of the *prouteCoreRingRTYLo* environment variable.
`?areaXHi` *`f_areaXHi`*	Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. If no value is specified, uses the value of the *prouteCoreRingRTXHi* environment variable.
`?areaYHi` *`f_areaYHi`*	Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. If no value is specified, uses the value of the *prouteCoreRingRTYHi* environment variable.
`?horiLayer` *`t_horiLayer`*	The horizontal metal layer to use to route the core ring. Value is a layer string. If no value is specified, uses the value of the *prouteCoreRingHorizLayer* environment variable.
`?vertLayer` *`t_vertLayer`*	The vertical metal layer to use to route the core ring. Value is a layer string. If no value is specified, uses the value of the *prouteCoreRingVertLayer* environment variable.

`?latticeStyle` *`g_latticeStyle`*

	Extend duplicate net segments to or from a lattice. Values are `t` or `nil`. If set to `nil`, the segments are concentric. If no value is specified, uses the value of the *prouteCoreRingLatticeStyle* environment variable.
`?netClearance` *f_netClearance*	The minimum clearance between any two power nets. Value is a positive, non-zero floating number. If no value is specified, uses the value of the *prouteCoreRingNetClearance* environment variable.
`?netWidth` *f_netWidth*	The total width of the nets to be routed. Value is a positive, non-zero floating number. If no value is specified, uses the value of the *prouteCoreRingNetWidth* environment variable.

Value Returned

`t`	The core ring scheme was created successfully.
`nil`	An error occurred and the core ring scheme was not created.

Examples

```
rteCreateCoreRingScheme(
?name 'schemeOne
?ringAtCenter nil
?relativeTo 'outsideCore
?coreClearance 2.000000
?padClearance 0.000000
?inAreaClearance 0.000000
?outAreaClearance 0.000000
?areaXLo 0.000000
?areaYLo 0.000000
?areaXHi 0.000000
?areaYHi 0.000000
?horiLayer "Metal3"
?vertLayer "Metal6"
?latticeStyle nil
?netClearance 2.000000
?netWidth 2.000000
)
```

Related Topics

Space-based Router Functions

rteCreatePadRingScheme

```
rteCreatePadRingScheme(
     [ ?name s_name ]
     [ ?allPinLayers g_allPinLayers ]
     [ ?pinLayers l_pinLayers ]
     [ ?railPins g_railPins ]
     [ ?edgePins g_edgePins ]
     ) ;
     => t / nil
```

Description

Creates a pad ring routing scheme.

Arguments

`?name` *`s_name`*	Name of the scheme. Value is a quoted symbol.
`?allPinLayers` *`g_allPinLayers`*	Connect pad pins on all routing layers. Values are `t` or `nil`. If `nil`, the router connects pad pins specified by the `?pinLayers` argument. If no value is specified, uses the value of the *routePadRingAllLayers* environment variable.
`?pinLayers` *`l_pinLayers`*	The layers to use to connect pad pins. Value is a list of layer strings. Separate layers with spaces. The `?allRouteLayers` argument must be set to `nil` to use this argument. If no value is specified, uses the value of the *routePadRingLayers* environment variable.
`?railPins` *`g_railPins`*	Route to rail pins on the pads. Values are `t` or `nil`. If no value is specified, uses the value of the *routePadRingRailPins* environment variable.
`?edgePins` *`g_edgePins`*	Route to the pin edge. Values are `t` or `nil`. If no value is specified, uses the value of the *routePadRingEdgePins* environment variable.

Value Returned

`t`	The pad ring scheme was created successfully.
`nil`	An error occurred and the pad ring scheme was not created.

Examples

```
rteCreatePadRingScheme(
?name 'schemeOne
?allPinLayers nil
?pinLayers '("Metal4" "Metal5" "Metal6")
?railPins t
?edgePins t
)
```

Related Topics

Space-based Router Functions

rteCreatePinToTrunkScheme

```
rteCreatePinToTrunkScheme(
     [ ?name s_name ]
     [ ?allPinLayers g_allPinLayers ]
     [ ?pinLayers l_pinLayers ]
     [ ?useTrunkLayer g_useTrunkLayer ]
     [ ?trunkLayer t_trunkLayer ]
     [ ?minTrunkWidth f_minTrunkWidth ]
     [ ?minWireWidth f_minWireWidth ]
     [ ?maxWireWidth f_maxWireWidth ]
     ) ;
     => t / nil
```

Description

Creates a pin to trunk routing scheme. A trunk is usually a power net created by power routing commands such as pad ring, core ring, block ring, stripes, or cell rows.

Arguments

`?name` *`s_name`* Name of the scheme. Value is a quoted symbol.

`?allPinLayers` *`g_allPinLayers`*

Use all layers for pin to trunk routing. Values are `t` or `nil`. If `nil`, the router uses the layers specified by the `?pinLayers` argument.

If no value is specified, uses the value of the *routePinToTrunkAllLayers* environment variable.

`?pinLayers` *`l_pinLayers`* The layers to use for pin to trunk routing. Value is a list of layer strings. Separate layers with spaces.
The `?allPinLayers` argument must be set to `nil` to use this argument.

If no value is specified, uses the value of the *routePinToTrunkLayers* environment variable.

`?useTrunkLayer` *`g_useTrunkLayer`*

Use the trunk layer for routing . Values are `t` or `nil`. If set to `t`, `?trunkLayer` must contain a valid routing layer.

If no value is specified, uses the value of the *routePinToTrunkSpecTLayer* environment variable.

`?trunkLayer` *`t_trunkLayer`*

Use this layer as the trunk layer. Value is a layer string.

If no value is specified, uses the value of the *routePinToTrunkLayer* environment variable.

`?minTrunkWidth` *`f_minTrunkWidth`*

The minimum width of a target trunk for pin to trunk connection. Value is a positive, non zero, floating number.

If no value is specified, uses the value of the *routePinToTrunkMinTrunkWidth* environment variable.

`?minWireWidth` *`f_minWireWidth`*

The minimum wire width for pin to trunk connection. Value is a positive, non zero, floating number.

If no value is specified, uses the value of the *routePinToTrunkMinWireWidth* environment variable.

`?maxWireWidth` *`f_maxWireWidth`*

The maximum wire width for pin to trunk connection. Value is a positive, non zero, floating number.

If no value is specified, uses the value of the *routePinToTrunkMaxWireWidth* environment variable.

Value Returned

`t`	The pin to trunk scheme was created successfully.
`nil`	An error occurred and the pin to trunk scheme was not created.

Examples

```
rteCreatePinToTrunkScheme(
?name 'schemeOne
?allPinLayers nil
?pinLayers '("Metal5" "Metal6")
?useTrunkLayer t
?trunkLayer "Metal5"
?minTrunkWidth 3.000000
?minWireWidth 2.000000
?maxWireWidth 1.000000
)
```

Related Topics

Space-based Router Functions

rteCreateStripesScheme

```
rteCreateStripesScheme(
     [ ?name s_name ]
     [ ?horiStripes g_horiStripes ]
     [ ?yStep f_yStep ]
     [ ?horiLayers l_horiLayers ]
     [ ?vertStripes g_vertStripes ]
     [ ?xStep f_sStep ]
     [ ?vertLayers l_vertLayers ]
     [ ?pinClearance f_pinClearance ]
     [ ?netClearance f_netClearance ]
     [ ?netWidth f_netWidth ]
     [ ?minStripeWidth f_minStripeWidth ]
     [ ?offsetFrom s_offsetFrom ]
     [ ?leftOffset f_leftOffset ]
     [ ?bottomOffset f_bottomOffset ]
     [ ?useCenterLine g_useCenterLine ]
     ) ;
     => t / nil
```

Description

Creates a stripes routing scheme.

Arguments

?name *s_name*	Name of the scheme. Value is a quoted symbol.
?horiStripes *g_horiStripes*	Create horizontal stripes. Values are `t` or `nil`. If no value is specified, uses the value of the *prouteStripesHorizDir* environment variable.
?yStep *f_yStep*	When `?horiStripes` is set to `t`, the router uses this value to separate the horizontal stripes. Value is a positive, non zero floating number. If no value is specified, uses the value of the *prouteStripesYStep* environment variable.
?horiLayers *l_horiLayers*	

Use the specified horizontal layers to route the horizontal stripes. `?horiStripes` must be set to `t` for this argument to take affect. Value is a list of horizontal metal layers. Separate layers with spaces.

If no value is specified, uses the value of the *prouteStripesHorizLayers* environment variable.

`?vertStripes` *`g_vertStripes`*

Create Vertical stripes. Values are `t` or `nil`.

If no value is specified, uses the value of the *prouteStripesVertDir* environment variable.

`?xStep` *`f_xStep`*

When `?vertStripes` is set to `t`, the router uses this value to separate the vertical stripes. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteStripesXStep* environment variable.

`?vertLayers` *`l_vertLayers`*

Use the specified vertical layers to route the vertical stripes. `?vertStripes` must be set to `t` for this argument to take affect. Value is a list of vertical metal layers. Separate layers with spaces.

If no value is specified, uses the value of the *prouteStripesVertLayers* environment variable.

`?pinClearance` *`f_pinClearance`*

Use this clearance between signal pins and power nets. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteStripesPinClearance* environment variable.

`?netClearance` *`f_netClearance`*

Use this clearance between stripes. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteStripesNetClearance* environment variable.

`?netWidth` *`f_netWidth`*

Stripe width. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteStripesNetWidth* environment variable.

`?minStripeWidth` *`f_minStripeWidth`*

Minimum stripe length. Value is a positive, non zero floating number.

If no value is specified, uses the value of the *prouteStripesMinWidth* environment variable.

`?offsetFrom` *`s_offsetFrom`*

Stripes starting location. Values are the following:
‘designBoundary
‘origin
‘routingArea

If no value is specified, uses the value of the *prouteStripesOffsetFrom* environment variable.

`?leftOffset` *`f_leftOffset`*

When `?offsetFrom` is set to `'designBoundary`, use this value as the x location for the first vertical stripe relative to the left of the design area.

When `?offsetFrom` is set to `'origin`, use this value as the y location for the first horizontal stripe relative to the design origin.

When `?offsetFrom` is set to `'routingArea`, this value has no affect.

If no value is specified, uses the value of the *prouteStripesLeftOffset* environment variable.

`?bottomOffset` *`f_bottomOffset`*

When `?offsetFrom` is set to `'designBoundary`, use this value as the y location for the first horizontal stripe relative to the bottom of the design area.

When `?offsetFrom` is set to `'origin`, use this value as the x location for the first vertical stripe relative to the design origin.

When `?offsetFrom` is set to `'routingArea`, this value has no affect.

If no value is specified, uses the value of the *prouteStripesBottomOffset* environment variable.

`?useCenterLine` *`g_useCenterLine`*

Use the centerline of the stripes for offset measurement. Value is `t` or `nil`. If `nil`, the offset is measured from the left edge for the first vertical strip and from the bottom edge for the first horizontal stripe.

If no value is specified, uses the value of the *prouteStripesCenterLine* environment variable.

Value Returned

`t`	The stripes scheme was created successfully.
`nil`	An error occurred and the stripes scheme was not created.

Examples

```
rteCreateStripesScheme(
?name 'schemeOne
?horiStripes t
?yStep 2.000000
?horiLayers '("Metal3" "Metal7")
?vertStripes t
?xStep 3.000000
?vertLayers '("Metal4" "Metal6")
?pinClearance 1.000000
?netClearance 2.000000
?netWidth 3.000000
?minStripeLength 4.000000
?offsetFrom 'origin
?leftOffset 2.000000
?bottomOffset 3.000000
?useCenterLine t
)
```

Related Topics

Space-based Router Functions

rteCreateViasScheme

```
rteCreateViasScheme(
     [ ?name s_name ]
     [ ?layerRange g_layerRange ]
     [ ?minLayer t_minLayer ]
     [ ?maxLayer t_maxLayer ]
     [ ?cutArrayRule s_cutArrayRule ]
     [ ?cutArrayRows n_cutArrayRows ]
     [ ?cutArrayColumns n_cutArrayColumns ]
     ) ;
     => t / nil
```

Description

Creates a via insertion scheme.

Arguments

`?name` *s_name*	Name of the scheme. Value is a quoted symbol.
`?layerRange` *g_layerRange*	Use the layer range specified by the `?minLayer` and `?maxLayer` arguments. Values are `t` or `nil`. If `nil`, the router uses the top and bottom layers of the design as the min and max layers of the layer range. If no value is specified, uses the value of the *prouteViaInsertionLayerRange* environment variable.
`?minLayer` *t_minLayer*	Use this layer as the lowest metal layer for connection. The value is a layer string. If no value is specified, uses the value of the *prouteViaInsertionMinLayer* environment variable.
`?maxLayer` *t_maxLayer*	Use this layer as the highest metal layer for connection. The value is a layer string. If no value is specified, uses the value of the *prouteViaInsertionMaxLayer* environment variable.
`?cutArrayRule` *s_cutArrayRule*	

The cut via configuration shape. Possible values are:.
`'square`
`'rectangular`
`'arraySize`

If you specify `'arraySize`, you must also enter values for the `?cutArrayRows` and `?cutArrayColumns` arguments.

If no value is specified, uses the value of the *proute*ViaInsertion*CutArray* environment variable.

`?cutArrayRows` *`n_cutArrayRows`*

The number of cut rows. Value is an integer. The `?cutArrayRule` argument must be set to `'arraySize`, for this value to be in affect.

If no value is specified, uses the value of the *proute*ViaInsertion*Rows* environment variable.

`?cutArrayColumns` *`n_cutArrayColumns`*

The number of cut columns. Value is an integer. The `?cutArrayRule` argument must be set to `'arraySize`, for this value to be in affect.

If no value is specified, uses the value of the *proute*ViaInsertion*Columns* environment variable.

Value Returned

`t`	The vias scheme was created successfully.
`nil`	An error occurred and the vias scheme was not created.

Examples

```
rteCreateViasScheme(
?name 'schemeOne
?layerRange t
?minLayer "Metal3"
?maxLayer "Metal7"
?cutArrayRule 'arraySize
?cutArrayRows 2
?cutArrayColumns 1
)
```

Related Topics

Space-based Router Functions

rteDecomposeTrunks

```
rteDecomposeTrunks(
     ) ;
     => t / nil
```

Description

Converts pathSegs and vias in power trunks to normal, routed pathSegs and vias. The function uses the current cellview and the selected set.

Converts only pathSegs and vias that are in not routes. Creates new routes for each converted pathSeg and each converted via.

Converted pathSegs have a topology of `none`.

The output specifies the following.

- number of pathSegs selected and the number converted
- number of vias selected and the number converted
- total number of objects selected

Arguments

None

Value Returned

`t`	The function completed successfully with no problems encountered. However, it is possible that no pathSegs or vias were converted. Output messages give the conversion statistics.
`nil`	The function encountered problems. Often, a warning message displays.

Examples

Select the objects in the cellview to decompose.

```
rteDecomposeTrunks()
```

Related Topics

Space-based Router Functions

rteDeleteRoutedNets

```
rteDeleteRoutedNets(
     [ ?cv d_cvid ]
     [ ?keepPower g_keepPower ]
     ) ;
     => t / nil
```

Description

Deletes the routed paths of all nets in the design. Optionally, you can preserve routed power nets.

Arguments

`?cv` *`d_cvId`*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function.
`?keepPower` *`g_keepPower`*	Specifies to keep paths previously routed by the power router. Values are `t` or `nil`. If set to `t`, previously routed power is not deleted. If set to `nil` (default), power routes are deleted along with all routed paths in the design.

Value Returned

`t`	The routed paths were deleted successfully.
`nil`	The routed paths were not deleted successfully.

Examples

The following example preserves power routes while deleting all other routes in the design.

```
rteDeleteRoutedNets(
?cv (geGetRoutedCellView)
?keepPower t
)
```

Related Topics

Space-based Router Functions

rteFixAntenna

```
rteFixAntenna(
     [ ?cv d_cvid ]
     [ ?all g_all ]
     [ ?nets l_nets ]
     [ ?set g_set ]
     [ ?excludeNets l_excludeNets ]
     [ ?layerList l_layerList ]
     [ ?fullRulesChecking g_fullRulesChecking ]
     [ ?maxWiresToPush x_maxWiresToPush ]
     [ ?model l_models ]
     [ ?useDiodes g_useDiodes ]
     [ ?useJumpers g_useJumpers ]
     ) ;
     => t / nil
```

Description

Fixes process antenna violations for the entire design, a specific net, or nets in a set by inserting jumpers and/or diodes. You can choose to ignore specific nets.

To use this command, your design data must have been imported from LEF and DEF files that include process antenna keywords for setting values used by this function.

Arguments

`?cv` *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Default is the current cellview.
`?all` *g_all*	Performs process antenna checks on all nets. Values are `t` or `nil` (default). If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.
`?nets` *l_nets*	Performs process antenna checks on the specified list of nets. If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.

`?set` *g_set*	Performs process antenna checks on nets in the selected set. Use the Navigator to select the nets. Values are `t` or `nil` (default). If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.
`?excludeNets` *l_nets*	Excludes the specified list of nets during the check.
`?layerList` *l_layerList*	Specifies the layer(s) on which jumpers can be added. Only *validRoutingLayers* that are also in this list can be used. By default, jumpers can only be added on *validRoutingLayers*.
`?fullRulesChecking` *g_fullRulesChecking*	Values are `t` (default) or `nil`. Setting a value of `t` performs complete checks. Setting a value of `nil` results in faster, but less accurate checking.
`?maxWiresToPush` *x_maxWiresToPush*	Specifies the number of wires that can be pushed for a single jumper. Default is 2.
`?model` *l_models*	Specifies the model(s) to use: OXIDE1, OXIDE2, OXIDE3, OXIDE4. Default is OXIDE1.
`?useDiodes` *g_useDiodes*	Adds diodes to remove violations. Values are `t` or `nil` (default).
`?useJumpers` *g_useJumpers*	Adds jumpers to remove violations. Values are `t` (default) or `nil`.

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

Examples

```
rteFixAntenna(
?cv (geGetRoutedCellView)
?all t
?excludeNets list("net20" "net16")
?layerList list("metal3" "metal5")
```

```
?fullRulesChecking nil
?maxWiresToPush 3
?models list("OXIDE2" "OXIDE3")
?useDiodes t
)
```

Related Topics

Space-based Router Functions

rteFixViolations

```
rteFixViolations(
     [ ?cv d_cvID ]
     [ ?closeOpens g_closeOpens ]
     [ ?excludeNet t_excludeNet ]
     [ ?excludePowerGround g_excludePowerGround ]
     [ ?fixCrossing g_fixCrossing ]
     [ ?fixMfgGrid g_fixMfgGrid ]
     [ ?fixMinArea g_fixMinArea ]
     [ ?fixMinAreaAtPins g_fixMinAreaAtPins ]
     [ ?fixMinEdge g_fixMinEdge ]
     [ ?fixMinEncArea g_fixMinEncArea ]
     [ ?fixMinWidth g_fixMinWidth ]
     [ ?fixNumCuts g_fixNumCuts ]
     [ ?fixPortShort g_fixPortShort ]
     [ ?fixRouteGrid g_fixRouteGrid ]
     [ ?reducePinWireNotches g_ReducePinWireNotches ]
     [ ?maximizeCuts g_maximizeCuts ]
     [ ?offsetVia g_offsetVia ]
     [ ?region l_region ]
     [ ?searchAndRepair g_searchAndRepair ]
     [ ?set g_set ]
     s_value
     ) ;
     => t / nil
```

Description

This function allows you to fix violations for a selected area or for an entire cellview.

Arguments

`?cv` *d_cvId*
: Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.

`?closeOpens` *g_closeOpens*
: Values are `t` or `nil`. If true (`t`), the detail router closes any remaining opens. If `nil`, no attempt is made to close the opens.

`?excludeNet` *t_excludeNet*
: A string of net names not to route. ?includeNet and ?excludeNet cannot both be specified.

`?excludePowerGround` *`g_excludePowerGround`*

Indicates whether to route power and ground nets. Values are `t` to exclude power and ground nets or `nil` to include power and ground nets.

If no value is specified, uses the value of the *excludeTypePowerGround* environment variable.

`?fixCrossing` *`g_fixCrossing`*

Directs the router to fix crosstalk violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesCrossing* environment variable

`?fixMfgGrid` *`g_fixMfgGrid`*

Directs the router to fix manufacturing grid violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesMfgGrid* environment variable

`?fixMinArea` *`g_fixMinArea`*

Directs the router to fix minimum area violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesMinArea* environment variable.

`?fixMinAreaAtPins` *`g_fixMinAreaAtPins`*

Directs the router to fix minimum area violations at pins by adding metal.

`?fixMinEdge` *`g_fixMinEdge`*

Directs the router to fix minimum edge violations by shifting or adding metal.

`?fixMinEncArea` *`g_fixMinEncArea`*

Directs the router to fix minimum enclosure area violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesMinEnclArea* environment variable

`?fixMinWidth` *`g_fixMinWidth`*

Directs the router to fix minimum width violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesMinWidth* environment variable

`?fixNumCuts` *`g_fixnumCuts`*

Directs the router to fix minimum number of cut violations by replacing the vias with more cuts.

`?fixPortShort` *`g_fixPortShort`*

Directs the router to fix port short violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesPortShort* environment variable

`?fixRouteGrid` *`g_fixRouteGrid`*

Directs the router to fix routing grid violations. Values are `t` or `nil` (default). If you include the ?region argument, the router fixes the specified errors within that area.

If no value is specified, uses the value of the *fixErrorsErrorTypesRGrid* environment variable

`?reducePinWireNotches` *`g_reducePinWireNotches`*

Directs the router to remove any small metal notches located near pins by shifting the wires slightly.

`?maximizeCuts` *`g_maximizeCuts`*

Directs the router to add more cuts to the vias without adding to the metal footprint.

`?offsetVias` *`g_offsetVias`*

	Allows vias to be offset while doing various fixes.
`?region` *`l_region`*	A list of floating point coordinates defining a region. If defined, all routing occurs within the region.
`?searchAndRepair` *`g_searchAndRepair`*	
	Directs the router to repair violations by localized re-routing.
`?set` *`g_set`*	Performs routing checks on nets in the selected set. Use the Navigator to select the nets.
	Values are `t` or `nil` (default).
	If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.
`s_value`	Specifies the value of the variable.

Value Returned

`t`	The variable was set successfully.
`nil`	The variable was not set successfully.

Examples

```
(rteFixViolation ?cv (geGetEditCellView)
                 ?fixCross t
                 ?fixPortShort t
                 ?region (list 10.0 10.0 20.0 20.0))
```

Related Topics

Space-based Router Functions

rteGeomAnd

```
rteGeomAnd(
     [ ?cv d_cvId ]
     [ ?lpp1 g_lpp1 ]
     [ ?lpp2 g_lpp2 ]
     [ ?outputLpp l_outputLpp ]
     [ ?blockedLayer l_blockedLayer ]
     [ ?polygons g_polygons ]
     [ ?region l_region ]
     ) ;
     => t / nil
```

Description

Generates new shapes that are common on two or more layer purposes.

Note: This function is available only in Layout XL and higher tiers.

Arguments

?cv *d_cvId*	Database ID of the cellview. Defaults to the current cellview.
?lpp1 *g_lpp1*	Specifies the first layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?lpp2 *g_lpp2*	Specifies the second layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?outputLpp *l_outputLpp*	Specifies the output layer and purpose for the generated shapes. Required. You must specify both the layer name and the layer purpose as a list or string. Examples: list("metal1" "drawing") "metal1 drawing"
?blockedLayer *l_blockedLayer*	

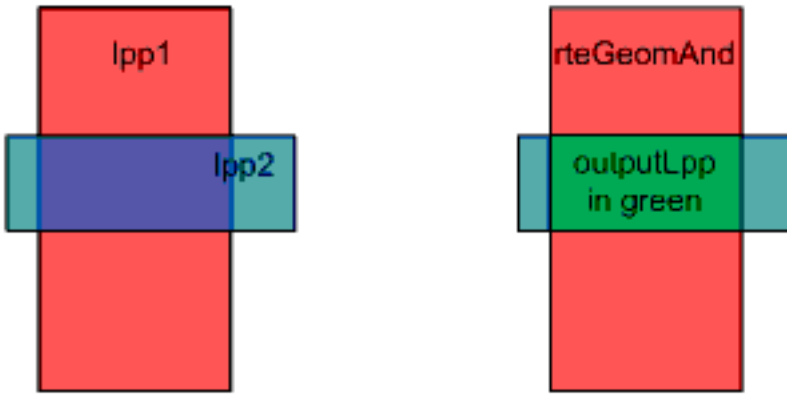
	Outputs blockages on the specified layer. Note: the ?blockedLayer and ?outputLpp arguments are mutually exclusive.
`?polygons` *`g_polygons`*	Generates output shapes as polygons, rather than individual rectangles. Values are `t` or `nil`. If set to `t` (true), the resulting shapes are polygons. If set to `nil` (default), the resulting shapes are rectangles.
`?region` *`l_region`*	A list specifying the lower-left and upper-right coordinates of the region on which the function should operate. If not specified, the entire design is included in the operation.

Value Returned

`t`	The operation completed successfully.
`nil`	The operation did not complete successfully.

Examples

The following is a graphical illustration of the `rteGeomAnd` function.



The following example generates new shapes on the `metal5/drawing` layer purpose that are common on the `metal2/drawing` and `metal1/pin` layer purposes.

```
rteGeomAnd(
?lpp1 list("metal2" "drawing")
?lpp2 list("metal1" "pin")
```

```
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

The following example is the same as the one above, but both layer purposes are included in the `?lpp1` argument.

```
rteGeomAnd(
?lpp1 list(
   list("metal2" "drawing")
   list("metal1" "pin"))
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

In the following example, `48` is the layer number representing `metal1`. Argument `?lpp1` includes all purposes for layer `metal1`. In argument `?lpp2`, the layer purpose pair is specified as a string. This is allowed, as the argument contains only one layer.

```
rteGeomAnd(
?lpp1 48
?lpp2 "metal2 drawing"
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
?polygons t
)
```

The following example outputs blockages on `metal4`.

```
rteGeomAnd(
?lpp1 list("poly")
?lpp2 list("oxide")
?blockedLayer "metal4")
```

Related Topics

Space-based Router Functions

rteGeomAndNot

```
rteGeomAndNot(
     [ ?cv d_cvId ]
     [ ?lpp1 g_lpp1 ]
     [ ?lpp2 g_lpp2 ]
     [ ?outputLpp l_outputLpp ]
     [ ?blockedLayer l_blockedLayer ]
     [ ?polygons g_polygons ]
     [ ?region l_region ]
     [ ?size1 f_size1 ]
     [ ?size2 f_size2 ]
     [ ?trimCorners g_trimCorners ]
     ) ;
     => t / nil
```

Description

Generates new shapes that exist on one layer purpose and do not overlap shapes on another layer purpose. You can optionally use sized shapes from one or both of the layer purposes. And you can choose to operate on a specific region or the entire design.

This function is available only in Layout XL and higher tiers.

Arguments

`?cv` *d_cvId*	Database ID of the cellview. Defaults to the current cellview.
`?lpp1` *g_lpp1*	Specifies the first layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
`?lpp2` *g_lpp2*	Specifies the second layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
`?outputLpp` *l_outputLpp*	Specifies the output layer and purpose for the generated shapes. Required. You must specify both the layer name and the layer purpose as a list or string. Examples: list("metal1" "drawing") "metal1 drawing"

`?blockedLayer` *`l_blockedLayer`*

Outputs blockages on the specified layer.

Note: the ?blockedLayer and ?outputLpp arguments are mutually exclusive.

`?polygons` *`g_polygons`* Generates output shapes as polygons, rather than individual rectangles. Values are `t` or `nil`.

If set to `t` (true), the resulting shapes are polygons.

If set to `nil` (default), the resulting shapes are rectangles.

`?region` *`l_region`* A list specifying the lower-left and upper-right coordinates of the region on which the function should operate.

If not specified, the entire design is included in the operation.

`?size1` *`f_size1`* Specifies the sizing amount in user units for shapes on the first layer purpose. Default is 0.0.

`?size2` *`f_size2`* Specifies the sizing amount in user units for shapes on the second layer purpose. Default is 0.0.

`?trimCorners` *`g_trimCorners`*

Specifies that corners be trimmed when shapes are resized.

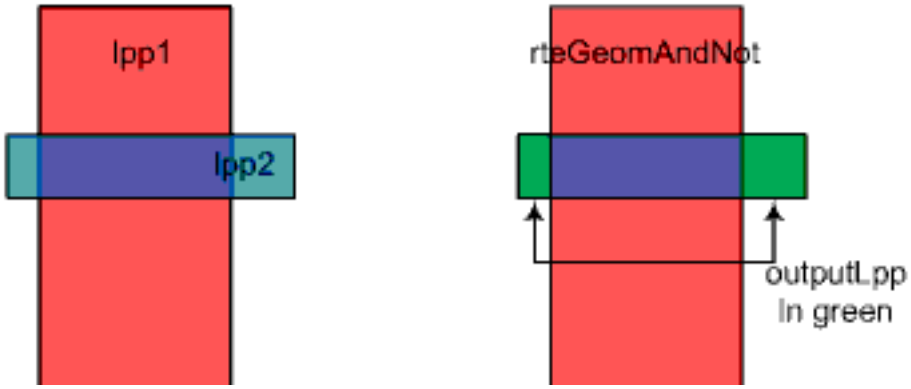
Values are `t` to trim corners or `nil` (default) to preserve corners.

Value Returned

`t`	The operation completed successfully.
`nil`	The operation did not complete successfully.

Examples

The following is a graphical illustration of the `rteGeomAndNot` function.



The following example generates new shapes on the `metal5/drawing` layer purpose that exist on `metal2/pin` and do not overlap shapes on `metal1/pin`.

```
rteGeomAndNot(
?lpp1 list("metal2" "drawing")
?lpp2 list("metal1" "pin")
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

The following example is the same as the one above, but both layer purposes are included in the `?lpp1` argument.

```
rteGeomAndNot(
?lpp1 list(
   list("metal2" "drawing")
   list("metal1" "pin"))
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

In the following example, 48 is the layer number representing `metal1`. Argument `?lpp1` includes all purposes for layer `metal1`. In argument `?lpp2`, the layer purpose pair is specified as a string. This is allowed, as the argument contains only one layer.

```
rteGeomAndNot(
?lpp1 48
?lpp2 "metal2 drawing"
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
?size1 0.5
?size2 0.5
?polygons t
?trimCorners t
)
```

Related Topics

Space-based Router Functions

rteGeomNot

```
rteGeomNot(
     [ ?cv d_cvId ]
     [ ?lpp1 g_lpp1 ]
     [ ?outputLpp l_outputLpp ]
     [ ?blockedLayer l_blockedLayer ]
     [ ?polygons g_polygons ]
     [ ?region l_region ]
     ) ;
     => t / nil
```

Description

Inverts the shapes on a layer purpose to generate new shapes.

This function is available only in Layout XL and higher tiers.

Arguments

?cv *d_cvId*	Database ID of the cellview. Defaults to the current cellview.
?lpp1 *g_lpp1*	Specifies the layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?outputLpp *l_outputLpp*	Specifies the output layer and purpose for the generated shapes. Required. You must specify both the layer name and the layer purpose as a list or string. Examples: list("metal1" "drawing") "metal1 drawing" **Note:** Lower level rectangles are mapped to blockage purpose as drawing is restricted to router changeable shapes. When the purpose blockage is used, the function is executed properly with the rectangle.
?blockedLayer *l_blockedLayer*	

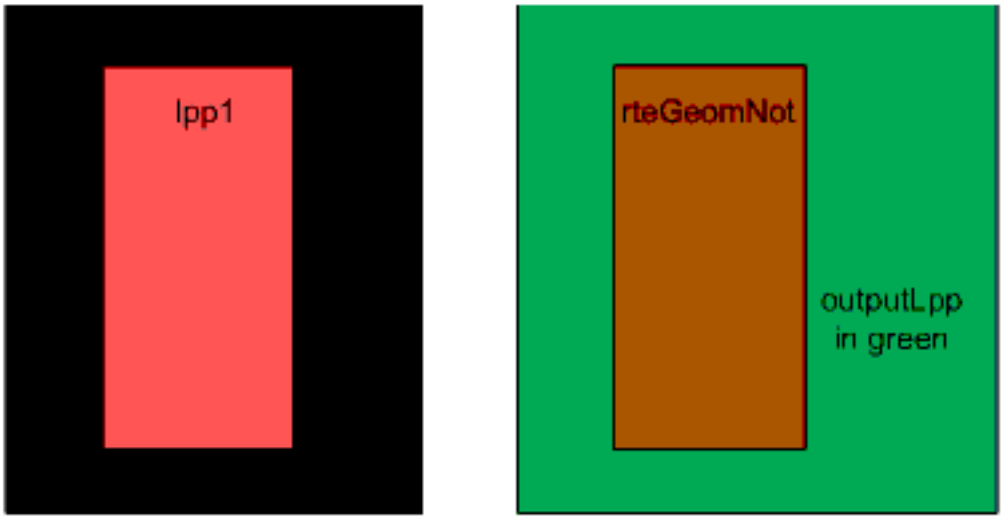
	Outputs blockages on the specified layer. Note: the ?blockedLayer and ?outputLpp arguments are mutually exclusive.
`?polygons` *`g_polygons`*	Generates output shapes as polygons, rather than individual rectangles. Values are `t` or `nil`. If set to `t` (true), the resulting shapes are polygons. If set to `nil` (default), the resulting shapes are rectangles.
`?region` *`l_region`*	A list specifying the lower-left and upper-right coordinates of the region on which the function should operate. If not specified, the entire design is included in the operation.

Value Returned

`t`	The operation completed successfully.
`nil`	The operation did not complete successfully.

Examples

The following is a graphical illustration of the `rteGeomNot` function.



The following example inverts the shapes on the `metal2/drawing` layer purpose and generates new shapes on the `metal5/drawing` layer purpose.

```
rteGeomNot(
?lpp1 list("metal2" "drawing")
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

In the following example, 48 is the layer number representing `metal1`. This example inverts all purposes for layer `metal1` and generates the new shapes on `metal5/drawing`. The `Metal5/drawing` layer purpose is expressed as a list, but it could also be expressed as a string, (`"metal5 drawing"`).

```
rteGeomNot(
?lpp1 48
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
?polygons t
)
```

Related Topics

Space-based Router Functions

rteGeomOr

```
rteGeomOr(
     [ ?cv d_cvId ]
     [ ?lpp1 g_lpp1 ]
     [ ?lpp2 g_lpp2 ]
     [ ?outputLpp l_outputLpp ]
     [ ?blockedLayer l_blockedLayer ]
     [ ?polygons g_polygons ]
     [ ?region l_region ]
     ) ;
     => t / nil
```

Description

Merges all shapes on the input layer purposes and generates new shapes on the output layer purpose.

This function is available only in Layout XL and higher tiers.

Arguments

?cv *d_cvId*	Database ID of the cellview. Defaults to the current cellview.
?lpp1 *g_lpp1*	Specifies the first layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?lpp2 *g_lpp2*	Specifies the second layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?outputLpp *l_outputLpp*	Specifies the output layer and purpose for the generated shapes. Required. You must specify both the layer name and the layer purpose as a list or string. Examples: list("metal1" "drawing") "metal1 drawing"
?blockedLayer *l_blockedLayer*	

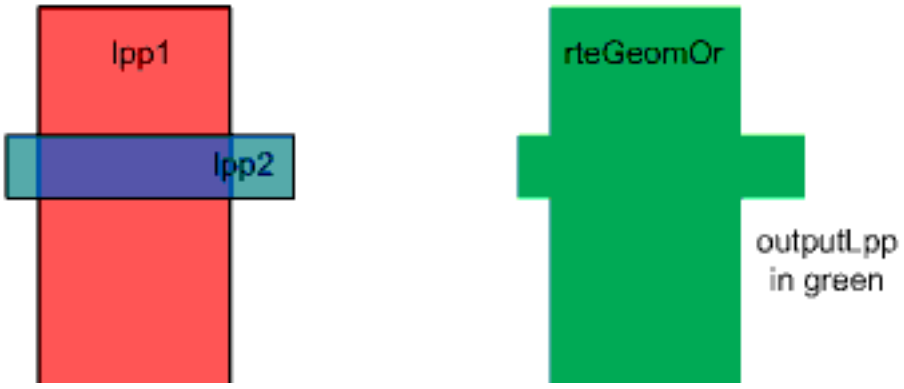
	Outputs blockages on the specified layer. Note: the ?blockedLayer and ?outputLpp arguments are mutually exclusive.
`?polygons` *`g_polygons`*	Generates output shapes as polygons, rather than individual rectangles. Values are `t` or `nil`. If set to `t` (true), the resulting shapes are polygons. If set to `nil` (default), the resulting shapes are rectangles.
`?region` *`l_region`*	A list specifying the lower-left and upper-right coordinates of the region on which the function should operate. If not specified, the entire design is included in the operation.

Value Returned

`t`	The operation completed successfully.
`nil`	The operation did not complete successfully.

Examples

The following is a graphical illustration of the `rteGeomOr` function.



The following example merges shapes on the `metal2/drawing` and `metal1/pin` layer purposes and generates new shapes on the `metal5/drawing` layer purpose.

```
rteGeomOr(
?lpp1 list("metal2" "drawing")
?lpp2 list("metal1" "pin")
```

```
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

The following example is the same as the one above, but both layer purposes are included in the `?lpp1` argument.

```
rteGeomOr(
?lpp1 list(
   list("metal2" "drawing")
   list("metal1" "pin"))
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

In the following example, 48 is the layer number representing `metal1`. Argument `?lpp1` includes all purposes for layer `metal1`. In argument `?lpp2`, the layer purpose pair is specified as a string. This is allowed, as the argument contains only one layer.

```
rteGeomOr(
?lpp1 48
?lpp2 "metal2 drawing"
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
?polygons t
)
```

Related Topics

Space-based Router Functions

rteGeomSize

```
rteGeomSize(
     [ ?cv d_cvId ]
     [ ?lpp1 g_lpp1 ]
     [ ?outputLpp l_outputLpp ]
     [ ?blockedLayer l_blockedLayer ]
     [ ?size f_size ]
     [ ?polygons g_polygons ]
     [ ?region l_region ]
     [ ?trimCorners g_trimCorners ]
     [ ?fixEdgesToRegion g_fixEdgesToRegion ]
     ) ;
     => t / nil
```

Description

Performs an oversize or undersize shapes on a specific layer purpose.

This function is available only in Layout XL and higher tiers.

Arguments

?cv *d_cvId*	Database ID of the cellview. Defaults to the current cellview.
?lpp1 *g_lpp1*	Specifies the first layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?outputLpp *l_outputLpp*	Specifies the output layer and purpose for the generated shapes. Required. You must specify both the layer name and the layer purpose as a list or string. Examples: list("metal1" "drawing") "metal1 drawing"
?blockedLayer *l_blockedLayer*	Outputs blockages on the specified layer. Note: the ?blockedLayer and ?outputLpp arguments are mutually exclusive.

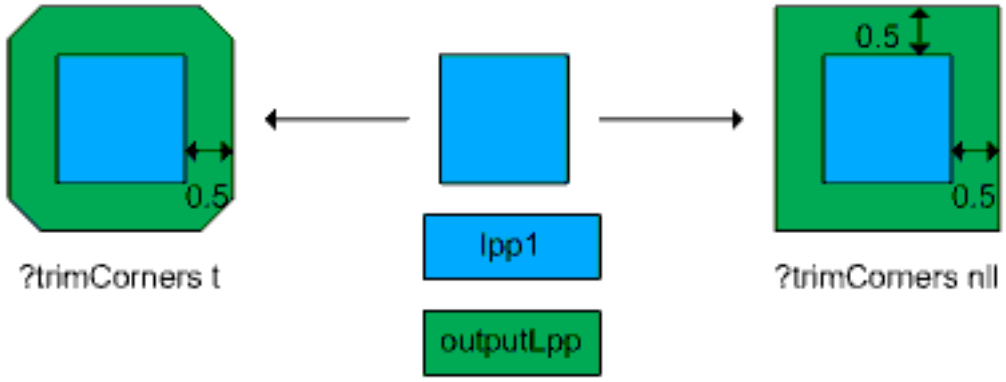
`?size` *`f_size`*	Specifies the sizing amount in user units for shapes on the first layer purpose. Required.
`?polygons` *`g_polygons`*	Generates output shapes as polygons, rather than individual rectangles. Values are `t` or `nil`. If set to `t` (true), the resulting shapes are polygons. If set to `nil` (default), the resulting shapes are rectangles.
`?region` *`l_region`*	A list specifying the lower-left and upper-right coordinates of the region on which the function should operate. If not specified, the entire design is included in the operation.
`?trimCorners` *`g_trimCorners`*	Specifies that corners be trimmed when shapes are resized. Values are `t` to trim corners or `nil` (default) to preserve corners.
`?fixEdgesToRegion` *`g_fixEdgesToRegion`*	Keeps edges fixed to region if a negative size is specified. Values are `t` (default if a region is specified) or `nil` (default is no region is specified).

Value Returned

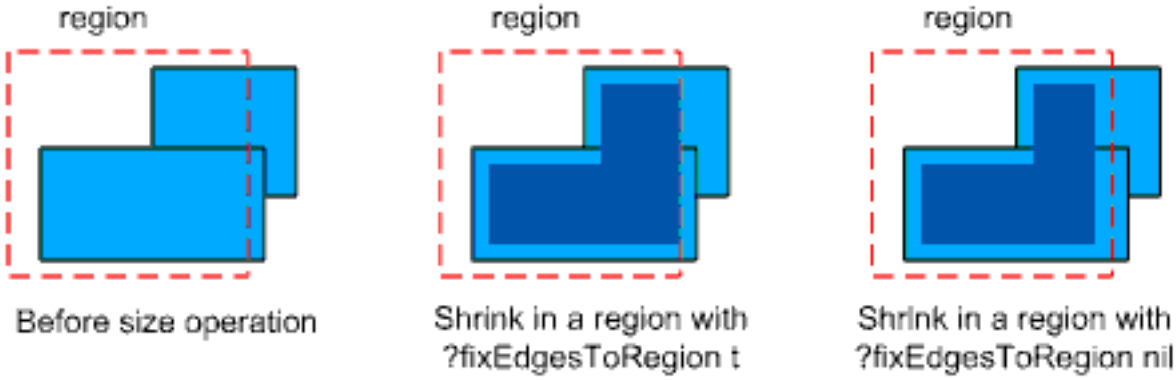
`t`	The operation completed successfully.
`nil`	The operation did not complete successfully.

Examples

The following illustrates the `?trimCorners` argument.



The following illustrates the `?fixEdgesToRegion` argument.



The following example generates new shapes on the `metal5/drawing` layer purpose by oversizing shapes on the `metal2/drawing` layer purpose by `1.5` user units.

```
rteGeomSize(
?lpp1 list("metal2" "drawing")
?outputLpp list("metal5" "drawing")
?size 1.5
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

In the following example, the layer purpose pair in argument `?lpp1` is specified as a string. This is allowed, as the argument contains only one layer. The example generates new shapes on the `metal6/drawing` layer purpose by undersizing shapes on the `metal1/drawing` layer purpose by `-2.5` user units.

```
rteGeomSize(
?lpp1 "metal1 drawing"
?outputLpp list("metal6" "drawing")
```

```
?size -2.5
?region list(2486.75 2850.75 2635.22 2946.52)
?polygons t
?trimCorners t
?fixEdgesToRegion t
)
```

Related Topics

Space-based Router Functions

rteGeomXor

```
rteGeomXor(
     [ ?cv d_cvId ]
     [ ?lpp1 g_lpp1 ]
     [ ?lpp2 g_lpp2 ]
     [ ?outputLpp l_outputLpp ]
     [ ?blockedLayer l_blockedLayer ]
     [ ?polygons g_polygons ]
     [ ?region l_region ]
     ) ;
     => t / nil
```

Description

Generates new shapes from shapes on either input layer purpose that do not overlap the other layer purpose.

This function is available only in Layout XL and higher tiers.

Arguments

?cv *d_cvId*	Database ID of the cellview. Defaults to the current cellview.
?lpp1 *g_lpp1*	Specifies the first layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.
?lpp2 *g_lpp2*	Specifies the second layer purpose pair(s). Required. See Specifying Input Layer Purposes for details.

?outputLpp *l_outputLpp*	Specifies the output layer and purpose for the generated shapes. Required.
	You must specify both the layer name and the layer purpose as a list or string.
	Examples:
	`list("metal1" "drawing")`
	`"metal1 drawing"`

?blockedLayer *l_blockedLayer*	Outputs blockages on the specified layer. Note: the ?blockedLayer and ?outputLpp arguments are mutually exclusive.

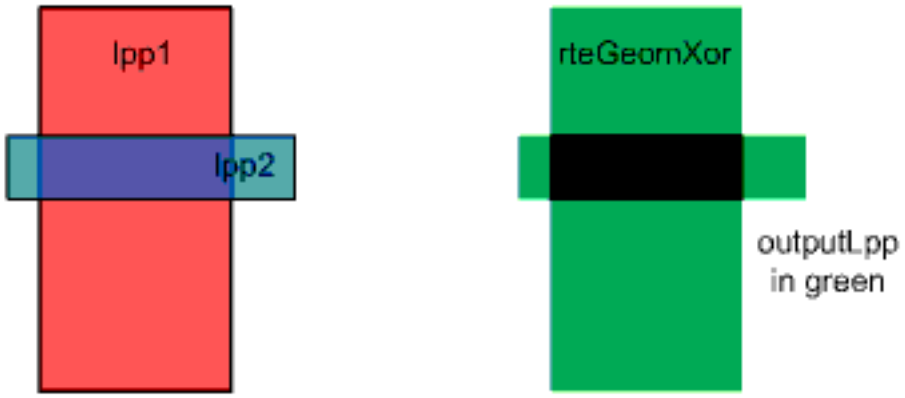
`?polygons` *`g_polygons`*	Generates output shapes as polygons, rather than individual rectangles. Values are `t` or `nil`. If set to `t` (true), the resulting shapes are polygons. If set to `nil` (default), the resulting shapes are rectangles.
`?region` *`l_region`*	A list specifying the lower-left and upper-right coordinates of the region on which the function should operate. If not specified, the entire design is included in the operation.

Value Returned

`t`	The operation completed successfully.
`nil`	The operation did not complete successfully.

Examples

The following is a graphical illustration of the `rteGeomXor` function.



The following example generates shapes on the `metal5/drawing` layer purpose from shapes on either the `metal2/drawing` or `metal1/pin` layer purpose that do not overlap the other.

```
rteGeomXor(
?lpp1 list("metal2" "drawing")
?lpp2 list("metal1" "pin")
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

The following example is the same as the one above, but both layer purposes are included in the `?lpp1` argument.

```
rteGeomXor(
?lpp1 list(
   list("metal2" "drawing")
   list("metal1" "pin"))
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
)
```

In the following example, `48` is the layer number representing `metal1`. Argument `?lpp1` includes all purposes for layer `metal1`. In argument `?lpp2`, the layer purpose pair is specified as a string. This is allowed, as the argument contains only one layer.

```
rteGeomXor(
?lpp1 48
?lpp2 "metal2 drawing"
?outputLpp list("metal5" "drawing")
?region list(2486.75 2850.75 2635.22 2946.52)
?polygons t
)
```

Specifying Input Layer Purposes

In the `rteGeom*` functions, you specify a layer set for arguments `?lpp1` and `?lpp2`. A layer set is comprised of a layer name, layer number, or layer purpose pairs. Express the layer set as a list or string.

You must specify layer purposes by name.

If you specify a layer name or number only, the function considers all layer purposes for that layer as input. To restrict layer purpose pairs, explicitly define the pairs.

If you specify only one layer, you can specify it as a string rather than a list.

Examples:

`list("metal1")`	Include all layer purpose pairs for metal1.
`list("metal2" "pin")`	Specifies the layer-purpose pair to use.
`50`	If 50 is the layer number representing metal3, include all layer purpose pairs for metal3. Since only one layer is included, the layer is specified as a string.
`list(53 "drawing")`	If 53 is the layer number representing metal4, specifies metal4/drawing as the layer purpose pair.

Related Topics

Space-based Router Functions

rteOptimizeRoute

```
rteOptimizeRoute(
     [ ?cv d_cvID ]
     [ ?checkAntenna g_checkAntenna ]
     [ ?detailCritic g_detailCritic ]
     [ ?excludeNet t_excludeNet ]
     [ ?excludePowerGround g_excludePowerGround ]
     [ ?reduceVias g_reduceVias ]
     [ ?region l_region ]
     [ ?set g_set ]
     [ ?useDoubleCutVias g_useDoubleCutVias ]
     ) ;
     => t / nil
```

Description

This function is used to optimize routing by performing refinement routing steps, such as reducing the vias, master vias, straighten wire.

Arguments

`?cv` *`d_cvId`*
: Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.

`?checkAntenna` *`d_checkAntenna`*
: Checks for process antenna violations for the entire design, specific nets, or nets in a set. You can choose to ignore specific nets.

`?detailCritic` *`g_detailCritic`*
: Directs the router to straighten wire paths. Values are `t` or `nil`.

  If no value is specified, uses the value of the *detailRouteCritic* environment variable.

`?excludeNet` *`t_excludeNet`*
: A string of net names not to route. ?includeNet and ?excludeNet cannot both be specified.

`?excludePowerGround` *`g_excludePowerGround`*

Indicates whether to route power and ground nets. Values are `t` to exclude power and ground nets or `nil` to include power and ground nets.

If no value is specified, uses the value of the *excludeTypePowerGround* environment variable.

`?reduceVias` *`g_reduceVias`*

If no value is specified, uses the value of the *optimizeRouteReduceVias* environment variable.

`?region` *`l_region`* — A list of floating point coordinates defining a region. If defined, all routing occurs within the region.

`?set` *`g_set`* — Performs routing checks on nets in the selected set. Use the Navigator to select the nets.

Values are `t` or `nil` (default).

If you specify the arguments ?all, ?nets, and ?set together, the preferred order is ?all ?nets ?set.

`?useDoubleCutVias` *`g_useDoubleCutVias`*

Replaces single-cut vias with double-cut vias.

Value Returned

`t`	The variable was set successfully.
`nil`	The variable was not set successfully.

Examples

```
(rteOptimizeRoute ?cv(geGetEditCellView)
                  ?critic t
                  ?reductVias t)
```

Related Topics

Space-based Router Functions

rtePowerRouteBlockRing

```
rtePowerRouteBlockRing(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?instances t_instances ]
     [ ?channels g_channel ]
     [ ?contour g_contour ]
     [ ?blockClearance f_blockClearance ]
     [ ?layers l_layers ]
     [ ?lattice g_lattice ]
     [ ?netClearance f_netClearance ]
     [ ?netWidth f_netWidth ]
     ) ;
     => t / nil
```

Description

Executes block ring routing for a layout cellview.

Arguments

?cv *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
?nets *l_nets*	A list of power nets. Required.
?instances *t_instances*	A list of block instances. Required.
?channels *g_channels*	Adds rails in the channels between block instances. Values are `t` or `nil`. If no value is specified, uses the value of the *prouteBlockRingChannels* environment variable.
?contour *g_contour*	Follow the contour of the block instances. Values are `t` or `nil`. If no value is specified, uses the value of the *prouteBlockRingContour* environment variable.
?blockClearance *f_blockClearance*	

	Separate the block rings and prBoundary of the block instance by the clearance value. The bounding box of the block is not considered, clearance is to the prBoundary. Value is a positive, non-zero floating number.
`?layers` *`l_layers`*	A list of two adjacent/orthogonal metal layers. Value is a layer string. Required.
`?lattice` *`g_lattice`*	Route using lattice style or concentric style. Values are `t` to indicate lattice style or `nil` to indicate concentric style. If no value is specified, uses the value of the *prouteBlockRingLattice* environment variable.
`?netClearance` *`f_netClearance`*	The minimum clearance between any two power nets. Value is a positive, non-zero floating number.
`?netWidth` *`f_netWidth`*	The total width of the nets to be routed. Value is a positive, non-zero floating number.

Value Returned

`t`	The block ring routed successfully.
`nil`	An error occurred and the block ring was not routed.

Examples

```
rtePowerRouteBlockRing(
?cv (geGetEditCellView)
?nets list("|VDD" "|GND")
?instances list("|I42")
?channels nil
?contour t
?blockClearance 3.0
?layers list("Metal3" "Metal4")
?lattice nil
?netClearance 0.14
?netWidth 3.0
)
```

Related Topics

Space-based Router Functions

rtePowerRouteCellRow

```
rtePowerRouteCellRow(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?layers l_layers ]
     [ ?routingArea l_routingArea ]
     [ ?rowEnd g_rowEnd ]
     [ ?extend g_extend ]
     ) ;
     => t / nil
```

Description

Executes cell row routing for a layout cellview.

Arguments

`?cv` *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
`?nets` *l_nets*	A list of Power nets.
`?layers` *l_layers*	A list of metal layers for routing.
`?routingArea` *l_routingArea*	List of two coordinates defining the area to route in the following format: `list(` *f_xlo f_ylo f_xhi f_yhi* `)`
`?rowEnd` *g_rowEnd*	Route standard cells to the end of defined rows. Values are `t` or `nil`. If `nil`, standard cells are routed to the last cell in the row. If no value is specified, the value of the *prouteCellRowRowEnd* environment variable is used.
`?extend` *g_extendToNearest*	Route the cell rows to the nearest power rail. Values are `t` or `nil`. If no value is specified, the value of the *prouteCellRowExtend* environment variable is used

Value Returned

`t`	The cell rows routed successfully.
`nil`	An error occurred and the cell rows were not routed.

Examples

```
rtePowerRouteCellRow(
?cv (geGetEditCellView)
?layers list("M2")
?routingArea list(-0.14 47.5 55.86 64.9)
?rowEnd t
?extend t
)
```

Related Topics

Space-based Router Functions

rtePowerRouteCoreRing

```
rtePowerRouteCoreRing(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?coreClearance f_coreClearance ]
     [ ?padClearance f_padClearance ]
     [ ?inAreaClearance f_inAreaClearance ]
     [ ?outAreaClearance f_outAreaClearance ]
     [ ?routingArea l_routingArea ]
     [ ?layers l_layers ]
     [ ?lattice g_lattice ]
     [ ?netClearance f_netClearance ]
     [ ?netWidth f_netWidth ]
     ) ;
     => t / nil
```

Description

Executes core ring power routing for a layout cellview.

Arguments

?cv *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
?nets *l_nets*	A list of power nets.
?coreClearance *f_coreClearance*	Clearance from the core ring to the core bounds. Value is a positive, non zero floating number.
?padClearance *f_padClearance*	Clearance from the core ring to the padstack instances. Value is a positive, non zero floating number.
?inAreaClearance *f_inAreaClearance*	Clearance from the core ring to the area specified by the ?routingArea argument. The core ring routes inside the routingArea. Value is a positive, non zero floating number.
?outAreaClearance *f_outAreaClearance*	Clearance from the core ring to the area specified by the ?routingArea argument. The core ring routes outside the routingArea. Value is a positive, non zero floating number.

`?routingArea` *`l_routingArea`*

List of two coordinates defining the area to route in the following format:

`list(` *`f_xlo f_ylo f_xhi f_yhi`* `)`

`?layers` *`l_layers`* — A list of two adjacent/orthogonal layers for routing.

`?lattice` *`g_lattice`* — Extend duplicate net segments to or from a lattice. Values are `t` or `nil`. If set to `nil`, the segments are concentric.

If not specified, the value of the *prouteCoreRingLattice* environment variable is used.

`?netClearance` *`f_netClearance`*

The minimum clearance between any two power nets. Value is a positive, non-zero floating number.

`?netWidth` *`f_netWidth`* — The total width of the nets to be routed. Value is a positive, non-zero floating number.

Value Returned

`t`	The core ring routed successfully.
`nil`	An error occurred and the core ring did not route.

Examples

```
rtePowerRouteCoreRing(
?cv (geGetEditCellView)
?nets list("MATCHB0" "MATCHB1")
?routingArea list(29.2 49.5 59.3 70.1)
?inAreaClearance 0.0
?layers list("metal2" "metal1")
?lattice nil
?netClearance 0.6
?netWidth 0.6
)
```

Related Topics

Space-based Router Functions

rtePowerRoutePadRing

```
rtePowerRoutePadRing(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?pads l_pads ]
     [ ?layers l_layers ]
     [ ?railPins g_railPins ]
     [ ?edgePins g_edgePins ]
     ) ;
     => t / nil
```

Description

Executes pad ring power routing for a layout cellview.

Arguments

?cv *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
?nets *l_nets*	List of power nets to route.
?pads *l_pads*	List of pad instances.
?layers *l_layers*	List of layers on which to route the pad ring.
?railPins *g_railPins*	Route to rail pins on the pads. Values are `t` or `nil`. If no value is specified, uses the value of the *proutePadRingRailPins* environment variable.
?edgePins *g_edgePins*	Route to the pin edge. Values are `t` or `nil`. If no value is specified, uses the value of the *proutePadRingEdgePins* environment variable.

Value Returned

`t`	The pad ring routed successfully.
`nil`	The pad ring did not route successfully.

Examples

```
rtePowerRoutePadRing(
?cv (geGetEditCellView)
```

```
?nets list("gnda!" "gndb!" "gndc!" "gndd!" "gnde5a!")
?pads nil
?layers nil
?railPins t
?edgePins t
)
```

Related Topics

Space-based Router Functions

rtePowerRoutePinToTrunk

```
rtePowerRoutePinToTrunk(
     [ ?cv d_cvId ]
     [ ?instances l_instances ]
     [ ?layers l_layers ]
     [ ?nets l_nets ]
     [ ?trunkLayer t_trunkLayer ]
     [ ?minTrunkWidth f_minTrunkWidth ]
     [ ?minWireWidth f_minWireWidth ]
     [ ?maxWireWidth f_maxWireWidth ]
     ) ;
     => t / nil
```

Description

Executes pin to trunk routing for a layout cellview. A trunk is usually a power net created by power routing commands such as pad ring, core ring, block ring, stripes, or cell rows.

Arguments

`?cv d_cvId`	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
`?instances l_instances`	List of instances to which pin to trunk routing should be applied.
`?layers l_layers`	List of layers on which this pin to trunk routing operation applies.
`?nets l_nets`	List of power nets. Required.
`?trunkLayer t_trunkLayer`	Use this layer as the trunk layer. Value is a layer string.
`?minTrunkWidth f_minTrunkWidth`	The minimum width of a target trunk for pin to trunk connection. Value is a positive, non zero, floating number. Required.
`?minWireWidth f_minWireWidth`	The minimum wire width for pin to trunk connection. Value is a positive, non zero, floating number. Required.
`?maxWireWidth f_maxWireWidth`	

	The maximum wire width for pin to trunk connection. Value is a positive, non zero, floating number. Required.

Value Returned

`t`	The pin to trunk routing completed successfully.
`nil`	The pin to trunk routing did not complete successfully.

Examples

```
rtePowerRoutingPinToTrunk(
?cv (geGetEditCellView)
?nets list("|GND" "|VDD")
?trunkLayer "metal1"
?minTrunkWidth 0.46scheme
?minWireWidth 2.0
?maxWireWidth 2.0
)
```

Related Topics

Space-based Router Functions

rtePowerRouteStripes

```
rtePowerRouteStripes(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?routingArea l_routingArea ]
     [ ?xStep f_sStep ]
     [ ?yStep f_yStep ]
     [ ?layers l_layers ]
     [ ?pinClearance f_pinClearance ]
     [ ?netClearance f_netClearance ]
     [ ?netWidth f_netWidth ]
     [ ?minLength f_minLength ]
     [ ?leftOffset f_leftOffset ]
     [ ?bottomOffset f_bottomOffset ]
     [ ?xOffset f_xOffset ]
     [ ?yOffset f_yOffset ]
     [ ?centerLine g_centerLine ]
     ) ;
     => t / nil
```

Description

Executes stripes power routing for a layout cellview.

Arguments

`?cv` *`d_cvId`*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
`?nets` *`l_nets`*	List of power nets. Required.
`?routingArea` *`l_routingArea`*	List of two coordinates defining the area to route in the following format: `list(` *`f_xlo f_ylo f_xhi f_yhi`* `)` Do not set `?xOffset` and `?yOffset` if you specify `?leftOffset`, `?bottomOffset`, and `?routingArea`.
`?xStep` *`f_xStep`*	The router uses this value to separate the vertical stripes. Value is a positive, non zero floating number. You cannot specify xStep and yStep at the same time, as it affects how the function determines direction.

`?yStep` *`f_yStep`*
: The router uses this value to separate the horizontal stripes. .Value is a positive, non zero floating number.

  You cannot specify xStep and yStep at the same time, as it affects how the function determines direction.

`?layers` *`l_layers`*
: List of metal layers on which to route. Required.

`?pinClearance` *`f_pinClearance`*
: Use this clearance between signal pins and power nets. Value is a positive, non zero floating number.

`?netClearance` *`f_netClearance`*
: Use this clearance between stripes. Value is a positive, non zero floating number.

`?netWidth` *`f_netWidth`*
: Stripe width. Value is a positive, non zero floating number.

`?minLength` *`f_minLength`*
: Minimum stripe length. Value is a positive, non zero floating number.

`?leftOffset` *`f_leftOffset`*
: The x location of the first vertical stripe relative to the left boundary. `?routingArea` must be set for this argument to be effective. Value is a non-zero floating number.
  Do not set `?leftOffset`, `?bottomOffset`, and `?routingArea` if you specify `?xOffset` and `?yOffset`.

`?bottomOffset` *`f_bottomOffset`*
: The y location of the first horizontal stripe relative to the bottom boundary. `?routingArea` must be set for this argument to be effective. Value is a non-zero floating number.
  Do not set `?leftOffset, ?bottomOffset,` and `?routingArea` if you specify `?xOffset and ?yOffset`.

`?xOffset` *`s_xOffset`*
: The x offset from the origin of the design to the first vertical stripe. Value is a non-zero floating number.
  Do not set `?xOffset` and `?yOffset` if you specify `?leftOffset`, `?bottomOffset`, and `?routingArea`.

`?yOffset` *`s_yOffset`*	The y offset from the origin of the design to the first horizontal stripe. Value is a non-zero floating number. Do not set `?xOffset` and `?yOffset` if you specify `?leftOffset, ?bottomOffset,` and `?routingArea`.
`?centerLine` *`g_centerLine`*	Use the centerline of the stripes for offset measurement. Value is `t` or `nil`. If `nil`, the offset is measured from the left edge for the first vertical strip and from the bottom edge for the first horizontal stripe. If no value is specified, uses the value of the *prouteStripesCenterLine* environment variable.

Value Returned

`t`	The stripes were routed successfully.
`nil`	The stripes were not routed successfully.

Examples

```
rtePowerRouteStripes(
?cv (geGetEditCellView)
?nets list("|GND" "|VDD")
?routingArea list(412.84 1510.24 981.815 1682.005)
?xStep 8.0
?yStep 8.0
?layers list("Metal5" "Metal6")
?pinClearance 0.14
?netClearance 2.0
?netWidth 2.0
?minLength 0.14
?leftOffset 0.0
?bottomOffset 0.0
?centerline nil
)
```

Related Topics

Space-based Router Functions

rtePowerRouteTieShield

```
rtePowerRouteTieShield(
     [ ?cv d_cvId ]
     [ ?shieldTieFreq n_shieldTieFreq ]
     [ ?coaxTieFreq n_coaxTieFreq ]
     ) ;
     => t / nil
```

Description

Ties shield wires to the shield nets in the entire design. Executed after creating shield nets either automatically or interactively.

Arguments

`?cv` *`d_cvId`*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
`?shieldTieFreq` *`n_shieldTieFreq`*	Specifies the maximum distance between ties that must be inserted to tie the new shield wires to their respective shield nets. Default is -1.0, which means unspecified.
`?coaxTieFreq` *`n_coaxTieFreq`*	Specifies the maximum distance between ties that must be inserted to tie tandem shield wires and parallel shield wires for coaxial shielding. Default is -1.0, which means unspecified.

Value Returned

`t`	Shield nets were tied successfully.
`nil`	Shield nets were not tied successfully.

Examples

```
rtePowerRouteTieShield(
?cv (geGetEditCellView)
?shieldTieFreq 1.5
```

```
?coaxTieFreq 1.0
)
```

Related Topics

Space-based Router Functions

rtePowerRouteTrimStripes

```
rtePowerRouteTrimStripes(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?layers l_layers ]
     [ ?includeStraps g_includeStraps ]
     [ ?undoable g_undoable ]
     ) ;
     => t / nil
```

Description

Trim existing stripes created by the rtePowerRouteStripes function.

This function is commonly executed after running rtePowerRouteStripes. You can run the function either before or after rtePowerRouteViaInsertion.

Arguments

`?cv` *`d_cvId`*	Database ID of the cellview. If not given, the cellview is automatically obtained using the *geGetEditCellView* SKILL function.
`?nets` *`l_nets`*	List of stripe nets to trim. Required.
`?layers` *`l_layers`*	List of metal layers on which to trim the nets. If not specified, stripes are trimmed on all layers.
`?includeStraps` *`g_includeStraps`*	Indicates whether to include straps in the trim operation. Values are `t` to include straps or `nil` (default) to not include straps.
`?undoable` *`g_undoable`*	Permits the function to be undone. Values are `t` to allow undo or `nil` to disallow undo. If not specified, the value of the *routeUndo* environment variable is used. If no value is specified and the *routeUndo* environment variable is not set, the argument defaults to `t`.

Value Returned

`t`	The stripes were trimmed successfully.
`nil`	The stripes were not trimmed successfully.

Examples

```
rtePowerRouteTrimStripes(
?cv (geGetEditCellView)
?nets list("|GND" "|VDD")
?layers list("Metal5" "Metal6")
?includeStraps t
?undoable nil
)
```

Related Topics

Space-based Router Functions

rtePowerRouteViaInsertion

```
rtePowerRouteViaInsertion(
     [ ?cv d_cvId ]
     [ ?nets l_nets ]
     [ ?instances l_instances ]
     [ ?layers l_layers ]
     [ ?minLayer t_minLayer ]
     [ ?maxLayer t_maxLayer ]
     [ ?rows n_rows ]
     [ ?columns n_columns ]
     [ ?squareCutArray g_squareCutArray ]
     [ ?useValidRoutingVias g_useValidRoutingVias ]
     ) ;
     => t / nil
```

Description

Executes via insertion between previously routed power rings, stripes, and cell rows.

Arguments

`?cv` *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
`?nets` *l_nets*	List of power nets. Required.
`?instances` *l_instances*	List of instances to which the via insertion applies.
`?layers` *l_layers*	List of layers to which the via insertion applies.
`?minLayer` *t_minLayer*	Lowest metal layer to be connected. Value is a layer string.
`?maxLayer` *t_maxLayer*	Highest metal layer to be connected. Value is a layer string.
`?rows` *n_rows*	Number of rows to use when inserting a via. Value is an integer. Use only when not specifying a `?squareCutArray`.
`?columns` *n_columns*	Number of columns to use when inserting a via. Value is an integer. Use only when not specifying a `?squareCutArray`.

`?squareCutArray` *g_squareCutArray*

Use a square via array. Values are `t` to use the square via array or `nil` to not use the square via array.
Use only when not specifying `?rows` and `?columns`.

`?useValidRoutingVias` *`g_useValidRoutingVias`*

Use vias listed in the `validVias` constraint. Values are `t` (default) to use valid routing vias or `nil`.

Value Returned

`t`	The vias were inserted successfully.
`nil`	The vias were not inserted successfully.

Examples

```
rtePowerRouteViaInsertion(
?cv (geGetEditCellView)
?nets list("|GND" "|VDD")
?minLayer "Metal1"
?maxLayer "Metal9"
?squareCutArray t
?useValidRoutingVias t
)
```

Related Topics

Space-based Router Functions

rteSearchAndRepair

```
rteSearchAndRepair(
     [ ?cv d_cvid ]
     [ ?closeOpens g_closeOpens ]
     [ ?excludeNet t_excludeNet ]
     [ ?excludeType t_excludeType ]
     [ ?set g_set ]
     [ ?region l_region ]
     [ ?fullRulesChecking g_fullRulesChecking ]
     ) ;
     => t / nil
```

Description

Searches for and fixes same net and different net spacing violations.

Arguments

?cv *d_cvId*	Database ID of the cellview. Can be obtained using the *geGetEditCellView* SKILL function. Required.
?closeOpens *g_closeOpens*	Values are `t` or `nil`. If true (`t`), the detail router closes any remaining opens. If `nil`, no attempt is made to close the opens.
?excludeNet *t_excludeNet*	A list of nets to exclude from processing.
?excludeType *t_excludeType*	Net types to exclude from processing. Values are `power`, `ground`, and/or `clock`.
?set *g_set*	Process only the set of nets selected in the Navigator Assistant.
?region *l_region*	A list specifying the lower-left and upper-right coordinates of the region on which the function should operate.

`?fullRulesChecking` *`g_fullRulesChecking`*

Perform all spacing related checks including merged shapes. Also checks minNumCuts and protrusionNumCuts.

Values are `t` or `nil`. Default is the value of the `routeFullRulesChecking` environment variable. The environment variable default is `t`.

Value Returned

`t`	Search and repair ran successfully.
`nil`	Search and repair did not run successfully.

Examples

```
rteSearchAndRepair(
?cv (geGetRoutedCellView)
?closeOpens t
?excludeNet "net20 net16"
?excludeType "power ground"
?set nil
?region list(10.0 10.0 50.0 50.5)
?fullRulesChecking nil
)
```

Related Topics

Space-based Router Functions

vsrDeletePreset

```
vsrDeletePreset(
     t_fileName
     [ ?directory t_directory ]
     [ ?path t_path ]
     [ ?confirmation g_confirmation ]
     ) ;
     => t / nil
```

Description

Deletes a preset file.

Arguments

t_fileName	The name of the preset file without `.preset` extension.
t_directory	The location of the preset file. It can have one of the following values: ■ `"."`: The current directory from where Virtuoso is run. The file is loaded from the `./.cadence/dfII/ia/presets` directory. This is the default value. ■ `"$HOME"`: The user home directory. The file is loaded from the `<user-home-directory>/.cadence/dfII/ia/presets` directory. ■ `"$CDS_PROJECT"`: The directory specified by the `CDS_PROJECT` shell environment variable.The file is loaded from the `$CDS_PROJECT/dfII/ia/presets` directory.
t_path	The location of the preset file. If both `?directory` and `?path` are specified, then the location specified in the `?path` is considered. However, if the `?path` is empty, the location specified in `?directory` is considered.
g_confirmation	Displays a confirmation dialog box before deleting the preset file. When set to `nil`, the preset file is deleted immediately without confirmation. Default is `t`.

Value Returned

`t`	The preset file is deleted.
`nil`	The function is unable to delete the preset file.

Examples

Deletes the preset file `commands` from the current run directory.

```
(vsrDeletePreset "commands")
```

Deletes preset file `M1M5Routing.preset` from the `$CDS_PROJECT/dfII/ia/presets` directory without any confirmation.

```
(vsrDeletePreset "M1M5Routing" ?directory "$CDS_PROJECT" ?confirmation nil)
```

Deletes preset file `M2M4Routing.preset` from the `./cadence/dfII/ia/presets` directory after confirming the delete operation.

```
(vsrDeletePreset "M2M4Routing" ?directory ".")
```

Related Topics

Space-based Router Functions

vsrLoadPreset

```
vsrLoadPreset(
     t_fileName
     [ ?directory t_directory ]
     [ ?execMode s_execMode ]
     [ ?path t_path ]
     ) ;
     => t / nil
```

Description

Loads a preset file to the Wire Assistant form and the Virtuoso Space-based Router Options form.

Arguments

t_fileName	The name of the preset file without `.preset` extension.
t_directory	The location of the preset file: ■ `"."`: The current directory from where Virtuoso is run. The file is loaded from the `./.cadence/dfII/ia/presets` directory. This is the default value. ■ `"$HOME"`: The user home directory. The file is loaded from the `<user-home-directory>/.cadence/dfII/ia/presets` directory. ■ `"$CDS_PROJECT"`: The directory specified by the `CDS_PROJECT` shell environment variable.The file is loaded from the `$CDS_PROJECT/dfII/ia/presets` directory.
t_path	The location of the preset file. If both `?directory` and `?path` are specified, then the location specified in the `?path` is considered. However, if the `?path` is empty, the location specified in `?directory` is considered.

s_execMode	The execution mode of the preset file. It can have one of the following two values, `'setting` and `'checking`. Default is `setting` . ■ `'setting` skips any preset entry that has an error and processes and loads all the entries that are error-free. ■ `'checking` processes the entries only if all the entries in the preset file are error free. If even one entry has an error, the function does not process any other entry, even if all others are error free.

Value Returned

`t`	The preset file is loaded.
`nil`	The function is unable to load the preset file.

Examples

The following example loads the preset file `commands` from the current run directory.

```
(vsrLoadPreset "commands")
```

Loads the preset file `M1M2Routing.preset` from the `$HOME/.cadence/dfII/ia/presets` directory but only if the preset file is error-free.

```
(vsrLoadPreset "M1M2Routing" ?directory "$HOME" ?execMode 'checking)
```

Loads the preset file `M3M5Routing.preset` from the `$CDS_PROJECT/dfII/ia/presets` directory. Valid settings are loaded, but any incorrect settings are skipped.

```
(vsrLoadPreset "M3M5Routing" ?directory "$CDS_PROJECT" ?execMode 'setting)
```

Related Topics

Space-based Router Functions

vsrSavePreset

```
vsrSavePreset(
     t_PresetLabel
     t_fileName
     [ ?directory t_directory ]
     [ ?path t_path ]
     [ ?tooltip t_tooltip ]
     [ ?description t_description ]
     [ ?iconFileName t_iconFileName ]
     [ ?iconText t_iconText ]
     [ ?location s_location ]
     [ ?mode s_mode ]
     [ ?envGroups l_envGroups ]
     ) ;
     => t / nil
```

Description

Saves the current settings to a preset file for later re-use.

Arguments

t_PresetLabel	The label of the preset file, saved as the `[label]:` keyword in the preset file. This preset label is used as an icon on the VSR Preset toolbar if both *?iconText* and *?iconFileName* are not specified or as an option name in the drop-down menu of the *VSR Preset Load* icon.
t_fileName	The name of the preset file without `.preset` extension.
t_directory	The location of the preset file. It can have one of the following values: ■ `"."`: The current directory from where Virtuoso is run. The file is loaded from the `./.cadence/dfII/ia/presets` directory. This is the default value. ■ `"$HOME"`: The user home directory. The file is loaded from the `<user-home-directory>/.cadence/dfII/ia/presets` directory. ■ `"$CDS_PROJECT"`: The directory specified by the `CDS_PROJECT` shell environment variable.The file is loaded from the `$CDS_PROJECT/dfII/ia/presets` directory.

t_path	The location of the preset file. If both `?directory` and `?path` are specified, then the location specified in the `?path` is considered. However, if the `?path` is empty, the location specified in `?directory` is considered.
t_tooltip	Saves as the `[tooltip]:` keyword in the preset file. This tooltip is displayed when the mouse cursor is placed on the preset icon. Default is empty string.
t_description	Saves as the `[description]:` keyword in the preset file. Default is empty string.
t_iconFileName	Saves as the `[icon]:` keyword in the preset file. This is the name of the icon file for a particular preset. Default is empty string.
t_iconText	Saves as the `[iconText]:` keyword in the preset file. This text appears on the created icon for a preset. Default is empty string.
s_location	The value of this option can be saved either as `'menu` or `'toolbar`. It is saved as the `[location]:` keyword in the preset file. Default is `'menu`.
s_mode	The mode controls how the Virtuoso Studio Design Environment is written. There are two modes: ■ `'all` saves current values of all VSR–related Virtuoso Studio Design Environment variables. This is the default mode. ■ `'edited` saves only the VSR–related Virtuoso Studio Design Environment variables that have values different from their default `.cdsenv` values.

l_envGroups	Controls which VSR–related Virtuoso Studio Design Environment group value is to be written. This is a list and can be one or more of the following values. The default value is `nil`, which means that all of the environment group values are written. ■ `'designSetup` the environment variables in the Design Setup section of the Virtuoso Space-based Router Options form. ■ `'automatic` all automatic routing (includes optimization, fix errors, delete) related environment variables. ■ `'p2t` all pin to trunk related environment variables ■ `'interactive` all wire editing related environment variables.

Value Returned

`t`	The preset file is saved.
`nil`	The function is unable to save the preset file.

Examples

Saves all the changed design setup and automatic environment variables to file `OptimizeRouting.preset` in the `$HOME/.cadence/dfII/ia/presets` directory with `Optimize Routing` as its label and `Opt Route` as the toolbar icon text.

```
(vsrSavePreset "Env Group" "envGroups" ?mode 'edited)

(vsrSavePreset "Optimize Routing"
                 "OptimizeRouting"
         ?directory "$HOME"
         ?iconText  "Opt Route"
         ?location  'toolbar
         ?mode       'edited
         ?envGroups  (list 'designSetup 'automatic)
  )
```

Related Topics

Space-based Router Functions

7

Floorplanner Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso Floorplanner.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- adpInitFloorplan
- adpnlSetEnv
- vfpAlignPins
- vfpAutoPin
- vfpCPHGenPhysicalHier
- vfpCPHGenPhysicalHierNoPropFile
- vfpCPHLoadFloorplanFile
- vfpCreateBoundaryPinsForSelectedShapes
- vfpHiPushPreRoutesDown
- vfpLoadPhysicalView
- vfpPinConnectivitySetting
- vfpPinSnapToEdge
- vfpPtAddPinResizeEstimator
- vfpPtGetPinResizeEstimators
- vfpPtLoadPinResizeFile
- vfpPtRemovePinResizeEstimator
- vfpPtRunPinResizeEstimator
- vfpPushPreRoutesDown

- vfpReportIOPadLocation
- vfpSbDefineObstruction
- vfpSbDeleteObstruction
- vfpSbEditIOPin
- vfpSbEditSoftBlockType
- vfpSbSetPolygonalBoundary
- vfpSbSetRectangularBoundary
- vpaOptimizePins

Introduction to Floorplanning SKILL Functions

Virtuoso Floorplanner SKILL functions are not interactive. They accept information as parameters and return status codes when specific tasks are complete.

Floorplanning place and route functions exist at two levels:

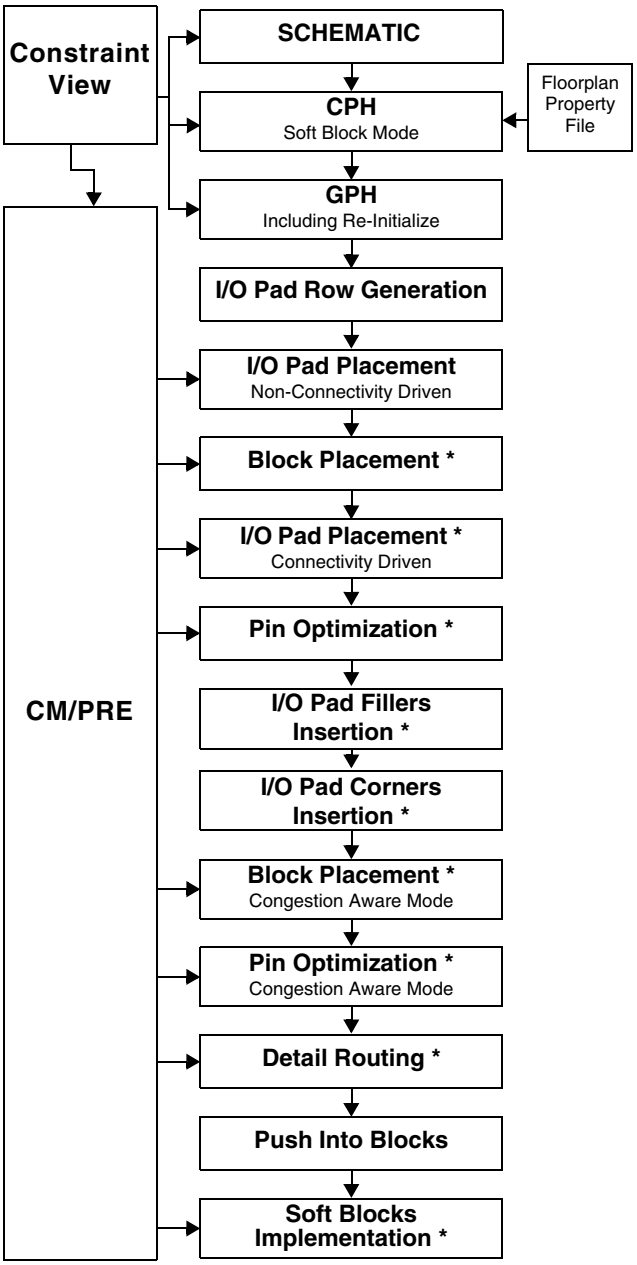
- High-level menu functions
- Low-level procedural access call functions

A high-level menu function is associated with each menu command or bindkey to control the user interaction. High-level menu functions provide the interface that collects and validates information before it is passed as parameters to one or more low-level procedural access functions performing the requested tasks.

This dual-level structure allows error checking and alternative action. Each low-level procedural access function returns status information following task completion, allowing the high-level menu functions to examine the status and to take alternative action if errors occur.

Cadence recommends that you use the floorplanning SKILL functions in the order depicted in the associated flowchart.

Floorplanning Flowchart Using Virtuoso Floorplanner SKILL Functions



Configure Physical Hierarchy
vfpCPHGenPhysicalHier generates a hierarchical floorplan.
vfpCPHLoadFloorplanFile populates the CPH form by using the property file.
vfpCPHGenPhysicalHierNoPropFile generates a physical configuration without a Floorplanning property file.

Initialize the Floorplan
adpInitFloorplan initializes the floorplan.

IO Planning and Placement
vfpReportIOPadLocation prints the IO pad report.
vfpSbDefineObstruction creates placement, routing, or other obstructions.
vfpSbDeleteObstruction deletes the specified obstruction.
vfpSbEditIOPin edits attributes of the pin that is associated with the specified termName.
vfpSbEditSoftBlockType edits the blockType of the specified soft block.
vfpSbSetPolygonalBoundary specifies the boundary attributes of soft blocks that are polygonal in shape.
vfpSbSetRectangularBoundary specifies the boundary vertices of the rectangular soft block.

Push Pre Routes
vfpHiPushPreRoutesDown Invokes the GUI to push pre-routes down.
vfpPushPreRoutesDown pushes the top-level power-structures of a cellview.

*** Optional manual editing like:**
=> Top-level PR boundary and pins manipulation
=> Soft blocks PR boundary and pins manipulation
=> Pin placer
=> Create feedthrough pins and soft pins
=> Pin alignment
=> Blockage creation
=> Wire editing

adpInitFloorplan

```
adpInitFloorplan(
     d_cv
     t_engine
     ?all g_all
     ?deleteClusterBndry g_deleteClusterBndry
     ?deleteCluster g_deleteCluster
     ?deleteBlockages g_deleteBlockages
     ?prsvPlacementBkg g_prsvPlacementBkg
     ?prsvRoutingBkg g_prsvRoutingBkg
     ?prsvAreaHaloBkg g_prsvAreaHaloBkg
     ?prsvFeedthruBkg g_prsvFeedthruBkg
     ?prsvFillBkg g_prsvFillBkg
     ?prsvPinBkg g_prsvPinBkg
     ?prsvScreenBkg g_prsvScreenBkg
     ?prsvSlotBkg g_prsvSlotBkg
     ?prsvWiringBkg g_prsvWiringBkg
     ?deleteWires g_deleteWires
     ?deleteRows g_deleteRows
     ?deleteFillers g_deleteFillers
     ?fillerPrefix t_fillerPrefix
     ?deleteShapes g_deleteShapes
     ?prsvCircle g_prsvCircle
     ?prsvDonut g_prsvDonut
     ?prsvEllipse g_prsvEllipse
     ?prsvPath g_prsvPath
     ?prsvPolygon g_prsvPolygon
     ?prsvRectangle g_prsvRectangle
     ?deleteBndry g_deleteBndry
     ?prsvAreaBndry g_prsvAreaBndry
     ?prsvSnapBndry g_prsvSnapBndry
     ?initGrid g_initGrid
     ?designBndry g_designBndry
     ?origin t_origin
     ?aspectRatio f_aspectRatio
     ?chipWidth f_chipWidth
     ?chipHeight f_chipHeight
     ?boundaryShapeType t_boundaryShapeType
     ?boundaryPoints g_boundaryPoints
     ?designArea f_designArea
     ?initInstances g_initInstances
     ?prsvIO g_prsvIO
     ?prsvStd g_prsvStd
     ?prsvMacro g_prsvMacro
     ?prsvCustomCell g_prsvCustomCell
     ?initPins g_initPins
     ?prsvPlacedPin g_prsvPlacedPin
     ?prsvFixedPin g_prsvFixedPin
     ?prsvLockedPin g_prsvLockedPin
     ?prsvFigGrp g_prsvFigGrp
```

```
?window g_window
)
=> t / nil
```

Description

Initializes the floorplan of a design based on the options you specify. Most of the options being keywords, you need to specify only those parameters that you want to control. The parameters that you do not specify use the default values.

Arguments

d_cv	Specifies the cellview identifier.
t_engine	Runs with any string value. This is now a redundant argument. It is only kept for backward portability reasons.

Note: For all of the following arguments, you must specify the keyword preceding the variable as shown in the syntax sample at the beginning of this function. However, for brevity, only the variables that you can specify are shown and described here.

g_all	When this option is set to t, reinitializes the design as if the *g_designBndry*, *g_deleteClusterBndry*,*g_deleteCluster*, *g_initGrid*, *g_initInstances*, *g_deleteWires*, and *g_deleteRows* arguments are set to `t`. The default is `t`.
?deleteClusterBndry *g_deleteClusterBndry*	Removes all cluster boundaries when this option is set to `t`. The default value is `t`.
?deleteCluster *g_deleteCluster*	Removes all clusters when this option is set to `t`. The default value is `t`.
?deleteBlockages *g_deleteBlockages*	Removes all the placement and routing blockages when this option is set to `t`. The default value is `t`.

?prsvPlacementBkg *g_prsvPlacementBkg*

Preserves the placement blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvRoutingBkg` *`g_prsvRoutingBkg`*

Preserves the routing blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvAreaHaloBkg` *`g_prsvAreaHaloBkg`*

Preserves the halo blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvFeedthruBkg` *`g_prsvFeedthruBkg`*

Preserves the feedthru blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvFillBkg` *`g_prsvFillBkg`*

Preserves the fill blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvPinBkg` *`g_prsvPinBkg`*

Preserves the pin blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvScreenBkg` *`g_prsvScreenBkg`*

Preserves the screen blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvSlotBkg` *`g_prsvSlotBkg`*

Preserves the slot blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?prsvWiringBkg` *`g_prsvWiringBkg`*

Preserves the wiring blockages when the `deleteBlockages` option is set to `t`. The default value is `nil`.

`?deleteWires` *`g_deleteWires`*

Removes all wires when this option is set to `t`. The default value is `t`.

`?deleteRows` *`g_deleteRows`*

Removes all rows when this option is set to `t`. The default value is `t`.

`?deleteFillers` *`g_deleteFillers`*

Removes the fillers with the specified prefix when the deleteFillers option is set to `t`. The default value is `t`.

`?fillerPrefix` *`t_fillerPrefix`*

<string argument>. Specifies the string for the filler prefix. The default value is set to 'Filler_'.

`?deleteShapes` *`g_deleteShapes`*

Removes all shapes when this option is set to `t`. The default value is `t`.

`?prsvCircle` *`g_prsvCircle`*

Preserves circles when deleteShapes option is set to `t`. The default value is `nil`.

`?prsvDonut` *`g_prsvDonut`* Preserves donut when deleteShapes option is set to `t`. The default value is `nil`.

`?prsvEllipse` *`g_prsvEllipse`*

Preserves ellipse when deleteShapes option is set to `t`. The default value is `nil`.

`?prsvPath` *`g_prsvPath`* Preserves path when deleteShapes option is set to `t`. The default value is `nil`.

`?prsvPolygon` *`g_prsvPolygon`*

Preserves polygon when deleteShapes option is set `t`. The default value is `nil`.

`?prsvRectangle` *`g_prsvRectangle`*

Preserves rectangles when deleteShapes option is set to `t`. The default value is `nil`.

`?deleteBndry` *`g_deleteBndry`*

Removes all area and snap boundaries when this option is set to `t`. The default value is `t`.

`?prsvAreaBndry` *`g_prsvAreaBndry`*

Preserves area boundary when deleteBndry option is set to `t`. The default value is `nil`.

`?prsvSnapBndry` *`g_prsvSnapBndry`*

Preserves snap boundary when deleteBndry option is set to `t`. The default value is `nil`.

`?initGrid` *`g_initGrid`* Initializes or re initializes routing, placement, and manufacturing grids when this option is set to `t`. The default value is `t`.

`?designBndry` *`g_designBndry`*

Resets the design size by estimation based on the aspect ratio or using the *`f_chipWidth`* and *`f_chipHeight`* options. The default value is `t`.

`?origin` *`t_origin`* Specifies the location of the design origin as '`center`' or '`lower left`'. The default value is '`lower left`'.

`?aspectRatio` *`f_aspectRatio`*

Specifies the design aspect ratio. You use this option when the `chipWidth` and the `chipHeight` are not defined. The default value is '1.0'.

`?chipWidth` *`f_chipWidth`* Specifies the design width in user units. When you specify this option, estimation is turned off. The default is '0.0' user units.

`?chipHeight` *`f_chipHeight`*

Specifies the design height in user units. When you specify this option, estimation is turned off. The default is '0.0' user units.

`?boundaryShapeType` *`t_boundaryShapeType`*

Specifies the shape of the boundary as rectangle or polygon. The default shape is '`rectangle`'.

`?boundaryPoints` *`g_boundaryPoints`*

Specifies the points of the PR boundary of the cellview that needs to be created. The default value is `nil`.

`?designArea` *`f_designArea`*

Specifies the area calculated by the user defined area estimator. The default value is '0.0'.

`?initInstances` *`g_initInstances`*

Sets the placement status of all placed instances to unplaced when this option is set to `t`. The default value is `t`.

`?prsvIO` *`g_prsvIO`* Preserves the placed IOs when the initInstance option is set to `t`. The default value is `nil`.

`?prsvStd` *`g_prsvStd`* Preserves the placement status of placed standard cells when the initInstance option is set to `t`. The default value is `nil`.

`?prsvMacro` *`g_prsvMacro`* Preserves the placement status of placed macro cells when the initInstance option is set to `t`. The default value is `nil`.

`?prsvCustomCell` *`g_prsvCustomCell`*

Preserves the placement status of placed custom cells when the initInstance option is set to `t`. The default value is `nil`.

`?initPins` *`g_initPins`* When this option is set to `t`, initializes the status of all pins to unplaced. The default value is `t`.

`?prsvPlacedPin` *`g_prsvPlacedPin`*

Preserves the placement status of placed pins when the initPins option is set to `t`. The default value is `nil`.

`?prsvFixedPin` *`g_prsvFixedPin`*

Preserves the placement status of fixed pins when the initPins option is set to `t`. The default value is `t`.

`?prsvLockedPin` *`g_prsvLockedPin`*

Preserves the placement status of locked pins when the initPins option is set to `t`. The default value is `t`.

`?prsvFigGrp` *`g_prsvFigGrp`*

Preserves the fig group and maintain the placement of its members. The default value is `t`.

`?window` *`g_window`* Specifies the window id with which the re-initialize form is attached. The default value is `nil`.

Value Returned

`t`	Returns `t` when adpInitFloorplan successfully initializes the design.
`nil`	Returns `nil` when `adpInitFloorplan` does not successfully initialize a design.

adpnlSetEnv

```
adpnlSetEnv(
     t_variable
     g_value
     [ ?domain s_domain ]
     )
     => t / nil
```

Description

Sets a variable in the specified domain. If the environment variable does not exist in the domain, a variable is created.

Arguments

t_variable	Name of the variable.
g_value	Value of the variable.
?domain *s_domain*	Domain in which the variable is present. By default it is `'adpEnvNL`.

Value Returned

`t`	Returns `t` if the variable value is set.
`nil`	Returns `nil` if the variable value is not set.

Example

```
adpnlSetEnv('adpEnvPinsPerNetThreshold 200 ?domain 'adpEnv)
```

In the above example, the `adpEnvPinsPerNetThreshold` variable is set in the 200 domain.

vfpAlignPins

```
vfpAlignPins(
     t_referencePinFigNameList
     t_targetPinfigNameList
     g_topCellViewId
     )
     => t / nil
```

Description

Aligns a list of target pins with respect to the position of a list of reference pins.

Arguments

t_referencePinFigNameList	List of reference pins. The following formats are supported to specify pin names: ■ The `Term:Pin:PinFig` format ■ The '.*' wildcard character and other regular expression constructs ■ Hierarchy delimiter '/' to specify level-1 pin names ■ Vector single and vector bit notations such as `A<0>, A<0:4>, A<2>:A<2>` in pin names
t_targetPinfigNameList	List of target pins. The following formats are supported to specify pin names: ■ The `Term:Pin:PinFig` format ■ The '.*' wildcard character and other regular expression constructs ■ Hierarchy delimiter '/' to specify level-1 pin names ■ Vector single and vector bit notations such as `A<0>, A<0:4>, A<2>:A<2>` in pin names
g_topCellViewId	ID of the top cellview

Value Returned

`t`	Pin alignment completed successfully.
`nil`	Pin alignment failed.

Examples

```
vfpAlignPins("A B C" "I0/A I0/B I0/C")
```

Here, the level-1 Pins; A,B,C; of Instance "IO" are aligned with reference to the top-level Pins; A,B,C.

```
vfpAlignPins("|IO/.*" ".*")
```

Here, all the top-level pins are aligned with reference to the level-1 pins of instance "IO".

```
vfpAlignPins("A:A:A B:B:B" "I1/A:A:A I2/B:B:B")
vfpAlignPins("IN<0:10>" "OUT<0:10>")
vfpAlignPins("|I2/OUT<0:15>" "OUT<0:15>")
vfpAlignPins("|I0/OUT<0:15>" "|I2/IN<0:15>")
vfpAlignPins("|I0/.* |I2/.*" "|I0/.* .*")
vfpAlignPins("|I0/IN<0:8> |I2/IN<0:9> |I2/OUT<0:12>" "|I0/OUT<0:12> IN_top<0:9> OUT<0:8>")
```

vfpAutoPin

```
vfpAutoPin(
     d_refView
     ?mode { cellview | selected }
     [ ?validRoutingLayer g_shapesOnRoutingLayersOnly ]
     [ ?purpose t_shapesOnSpecificPurpose ]
     [ ?schematicCheck g_schematicCheck ]
     [ ?deleteMode { auto | all } ]
     [ ?checkDuplicatePin g_checkDuplicatePin ]
     [ ?boundarySignalMode { single | multiple } ]
     [ ?buriedSignalMode { single | multiple } ]
     [ ?boundaryMultipleMode { topLayer | useLayer } ]
     [ ?buriedMultipleMode { topLayer | useLayer } ]
     [ ?boundarySignalPinLayers t_boundarySignalPinLayers ]
     [ ?buriedSignalPinLayers t_buriedSignalPinLayers ]
     [ ?boundaryPowerPinLayers t_boundaryPowerPinLayers ]
     [ ?buriedPowerPinLayers t_buriedPowerPinLayers ]
     [ ?boundaryPinSize t_boundaryPinSize ]
     [ ?buriedPinSize t_buriedPinSize ]
     [ ?targetLPP t_targetLPP ]
     [ ?pinConnectivity t_pinConnectivity ]
     [ ?createLabel g_createLabel ]
     [ ?enableColoring g_enableColoring ]
     [ ?createMetalShape g_createMetalShape ]
     [ ?metalShapeType { path | rectangle } ]
     [ ?hDepthCheck g_hDepthCheck ]
     [ ?startLevelValue x_startLevelValue ]
     [ ?createDeleteMode t_createDeleteMode ]
     [ ?preview g_preview ]
     [ ?customLengthsBoundaryShapePin t_customLengthsBoundaryShapePin ]
     [ ?customLengthsBuriedShapePin t_customLengthsBuriedShapePin ]
     )
     => t / nil
```

Description

Searches for net shapes in the specified window or cellview and automatically creates boundary and buried pins on these net shapes.

Arguments

d_refView	Specifies the cellView ID in which pins are to be created.
?mode { cellview|selected }	

Specifies the mode in which the command must be run.

`?validRoutingLayer` *`g_shapesOnRoutingLayersOnly`*

Restricts the search for net shapes to the routing layers. The default value is nil, where all layers are searched.

`?purpose` *`t_shapesOnSpecificPurpose`*

Filters the net shapes based on the specified purpose. The default value is `""`.

`?schematicCheck` *`g_schematicCheck`*

Specifies whether the tool must search for corresponding shapes in the schematic view before creating pins. The default value is `nil`.

`?deleteMode` { `auto` | `all` }

Specifies the mode in which existing pins must be deleted before new ones are created. The valid values are:

- `auto`: Specifies that only those pins that were created automatically by using the Auto Pin tool are to be deleted.
- `all`: Specifies that all pins in the current cellview are to be deleted.

`?checkDuplicatePin` *`g_checkDuplicatePin`*

Checks for the presence of any similar pins in the design. If a similar pin is found, then a duplicate pin is not created.

`?boundarySignalMode` { `single` | `multiple` }

Specifies the way in which boundary pins are created on shapes that are abutted to the PR boundary. The valid values are:

- `single`: Creates a single boundary pin on the highest metal layer.
- `multiple`: Creates as many boundary pins as the number of boundary shapes, in the same layers as the shapes.

The default value is `""`, which indicates that boundary signal pins are not created.

`?buriedSignalMode` { `single` | `multiple` }

Specifies the way in which buried pins are created on shapes that are completely inside the PR boundary, without any contact with the edges. The valid values are:

- `single`: Creates a single buried pin on the highest metal layer.
- `multiple`: Creates as many buried pins as the number of buried shapes, in the same layers as the shapes.

The default value is `""`, which indicates that buried signal pins are not created.

`?boundaryMultipleMode` { `topLayer` | `useLayer` }

Specifies the following layer selection options:

- `topLayer`: Creates multiple boundary pins on the highest metal layer.
- `useLayer`: Lets you specify the layers on which boundary pins are to be created. Use the `?boundarySignalPinLayers` argument to specify the layers.

The default value is `toplayer` when `boundarySignalMode` is set to `multiple`.

`?buriedMultipleMode` { `topLayer` | `useLayer` }

Specifies the following layer selection options:

- `topLayer`: Creates multiple buried pins on the highest metal layer.
- `useLayer`: Lets you specify the layers on which buried pins are to be created. Use the `?buriedSignalPinLayers` argument to specify the layers.

The default value is `toplayer` when `buriedSignalMode` is set to `multiple`.

`?boundarySignalPinLayers` *`t_boundarySignalPinLayers`*

Specifies the layers on which boundary signal pins must be generated when `?boundaryMultipleMode` is set to `useLayer`.

`?buriedSignalPinLayers` *`t_buriedSignalPinLayers`*

Specifies the layers on which buried signal pins must be generated when `?buriedMultipleMode` is set to `useLayer`.

`?boundaryPowerPinLayers` *`t_boundaryPowerPinLayers`*

Specifies the names of layers on which the boundary power pins can be created.

The default value is `""`, which indicates that no layers are specified, and therefore no pins are to be created on the boundary power net shape.

`?buriedPowerPinLayers` *`t_buriedPowerPinLayers`*

Specifies the names of layers on which the buried power pins can be created.

The default value is `""`, which indicates that no layers are specified, and therefore no pins are to be created on the buried power net shape.

`?boundaryPinSize` *`t_boundaryPinSize`*

Specifies the boundary pin size. The valid values are:

- `Square (Default)`: Creates square pins. Only rectangular and square path, pathsegs, and polygonal shapes are recognized for pin creation.
- `Whole Shape`: Creates pins of the same shape as the corresponding net shapes. For example, square, rectangle, L-shaped, and path pin shapes are supported.
- `Honor Min Length`: Honors the minLength constraint specified in the technology file. If the pin length is less than the minimum length specified in the constraint, the pin is resized to match minLength value during pin creation.
- `Custom Length`: Specifies a custom length value for a pin.

`?buriedPinSize` *`t_buriedPinSize`*

Specifies the buried pin size. The valid values are:

- `Square` (Default): Creates square pins. Only rectangular and square path, pathsegs, and polygonal shapes are recognized for pin creation.
- `Whole Shape`: Creates pins of the same shape as the corresponding net shapes. For example, square, rectangle, L-shaped, and path pin shapes are supported.
- `Honor Min Length`: Honors the minLength constraint specified in the technology file. If the pin length is less than the minimum length specified in the constraint, the pin is resized to match minLength value during pin creation.
- `Custom Length`: Specifies a custom length value for a pin.

`?targetLPP` *`t_targetLPP`*

Specifies the LPP on which pins must be generated. The valid values are:

- `sameLPP` (Default): Creates pins on the same LPPs as the original net shapes.
- `customPurpose`: Lets you specify a different purpose for the same layer.
- `customLPP`: Lets you specify a different LPP on which pins must be created.

`?pinConnectivity` *`t_pinConnectivity`*

Specifies the following pin connectivity options:

- `stronglyConnected` (Default): Specifies that the router must connect the pins internally within the device.
- `mustJoin`: Specifies that the router must connect the pins externally at a higher level.

`?createLabel` *`g_createLabel`*

Specifies whether text labels are created for pins. The default value is `nil`.

`?enableColoring` *`g_enableColoring`*

Specifies that the multiple patterning coloring engine be used to assign colors to the new pins based on their positions relative to the WSP tracks.

`?createMetalShape` *`g_createMetalShape`*

Creates a metal shape under each auto-created level-1 (soft block) pin. The metal shapes are assigned to the same layers as the parent pin and to the `drawing` purpose.

`?metalShapeType` { `path` | `rectangle` }

Specifies the dimensions of the metal shapes that are created under automatically created pins when `?createMetalShape` is set to `t`.

`?hDepthCheck` *`g_hDepthCheck`*

Enables hierarchical pin creation and pin promotion from a lower level to the top level.

`?startLevelValue` *`x_startLevelValue`*

Specifies the start level from which the tool has to start hierarchical pin creation and promotion.

`?createDeleteMode` *`t_createDeleteMode`*

Deletes all pins that were created by the Auto-Create Pins tool, creates pins hierarchically, and promotes them from a lower level to a higher level.

`?preview` *`g_preview`*

Displays a preview of how the settings would be applied to the design. The default value is `nil`.

`?customLengthsBoundaryShapePin` *`t_customLengthsBoundaryShapePin`*

Specifies a custom length value for a boundary shape pin.

`?customLengthsBuriedShapePin` *`t_customLengthsBuriedShapePin`*

Specifies a custom length value for a buried shape pin.

Value Returned

`t`	Pins were created as specified.
`nil`	Pins were not created.

Examples

The following examples show how `vfpAutoPin` can be used with different argument combinations.

Searches for net shapes in the selected nets and creates a single boundary signal pin on the highest metal layer.

```
vfpAutoPin(cvId "selected" ?boundarySignalMode "single")
```

Searches for net shapes with purpose `drawing` in the selected nets and creates multiple must-join boundary signal pins of the same shapes as the original net shapes.

```
vfpAutoPin(cvId "selected" ?purpose "drawing" ?boundarySignalMode "multiple"
?boundaryPinSize "whole" ?pinConnectivity "mustJoin" ?preview t)
```

Searches for net shapes on routing layers in the selected nets. Pins that were created using the auto pin tool are deleted before new ones are generated on the canvas. New boundary signal pins, boundary power pins, and buried power pins as per the specified arguments.

```
vfpAutoPin(cvId "selected" ?validRoutingLayer t ?deleteMode "auto"
?boundarySignalMode "multiple" ?buriedSignalMode "single" ?boundaryPowerPinLayers
"Metal1 Metal2" ?buriedPowerPinLayers "Meta1 Metal2" ?targetLPP "sameLPP"
?createLabel t)
```

Searches for net shapes in the current cellview and creates boundary power pins on the specified layers. Existing pins that were created using the auto pin tool are deleted before generating new pins on the canvas.

```
vfpAutoPin(cvId "selected" ?boundaryPowerPinLayers "Metal1 Metal2" ?deleteMode
"auto")
```

vfpCPHGenPhysicalHier

```
vfpCPHGenPhysicalHier(
     t_topLogicalLibName
     t_topLogicalCellName
     t_topLogicalViewName
     t_topPhysicalLibName
     t_topPhysicalCellName
     t_topPhysicalViewName
     t_FloorplanPropertyFileName
     )
     => t / nil
```

Description

Generates a hierarchical floorplan. It takes the Floorplan property file and the names of the logical and physical top cellview names as inputs. The top logical lib cellview name should match the names provided in the Floorplan property file.

Arguments

t_topLogicalLibName	Name of the top logical library
t_topLogicalCellName	Name of the top logical cell
t_topLogicalViewName	Name of the top logical view
t_topPhysicalLibName	Name of the top physical library
t_topPhysicalCellName	Name of the top physical cell
t_topPhysicalViewName	Name of the top physical view
t_FloorplanPropertyFileName	Name of the Floorplan property file

Value Returned

`t`	Returns `t` if the hierarchical floorplan is generated.
`nil`	Returns `nil` otherwise.

vfpCPHGenPhysicalHierNoPropFile

```
vfpCPHGenPhysicalHierNoPropFile(
     t_topLogicalLibName
     t_topLogicalCellName
     t_topLogicalViewName
     t_topPhysicalLibName
     t_topPhysicalCellName
     t_topPhysicalViewName
     )
     => t / nil
```

Description

Generates a physical configuration without a Floorplanning property file.

Arguments

t_topLogicalLibName	Name of the top logical library
t_topLogicalCellName	Name of the top logical cell
t_topLogicalViewName	Name of the top logical view
t_topPhysicalLibName	Name of the top physical library
t_topPhysicalCellName	Name of the top physical cell
t_topPhysicalViewName	Name of the top physical view

Value Returned

`t`	Returns `t` if the hierarchical floorplan is generated.
`nil`	Returns `nil` otherwise.

Example

```
vfpCPHGenPhysicalHierNoPropFile("design" "top_viewtest" "schematic" "design"
"top_viewtest" "layout_sav")
```

vfpCPHLoadFloorplanFile

```
vfpCPHLoadFloorplanFile(
     t_topLogicalLibName
     t_topLogicalCellName
     t_topLogicalViewName
     t_FloorplanPropertyFileName
     )
     => t / nil
```

Description

Populates the CPH form in Soft Block mode by using the Floorplan property file.

Arguments

t_topLogicalLibName	Name of the top logical library
t_topLogicalCellName	Name of the top logical cell
t_topLogicalViewName	Name of the top logical view
t_FloorplanPropertyFileName	Name of the Floorplan property file

Value Returned

`t`	Returns `t` if the CPH form is populated.
`nil`	Returns `nil` otherwise.

vfpCreateBoundaryPinsForSelectedShapes

```
vfpCreateBoundaryPinsForSelectedShapes(
     d_cellViewId
     [ n_distanceFromBoundary ]
     )
     => l_pinIds / nil
```

Description

Creates boundary pins on selected shapes.

Arguments

d_cellViewId	The cellview identifier
n_distanceFromBoundary	Distance between the pin shapes and the PRBoundary. If you do not specify the distance, then the default value of 10 is assigned.

Value Returned

l_pinIds	List of pin Ids generated
nil	The command was unsuccessful. No pins were created.

Example

```
pinIds = vfpCreateBoundaryPinsForSelectedShapes(cvId 8.0)
```

vfpHiPushPreRoutesDown

```
vfpHiPushPreRoutesDown(
     w_windowId
     )
```

Description

Invokes the GUI for pushing pre-routes down.

Arguments

w_windowId	Specifies the window id.

Value Returned

None

Example

```
vfpHiPushPreRoutesDown(window)
```

vfpLoadPhysicalView

```
vfpLoadPhysicalView(
     t_sourceLibName
     t_sourceCellName
     t_sourceViewName
     t_targetLibName
     t_targetCellName
     t_targetViewName
     [ ?updateInstances g_updateInstances ]
     [ ?updateInstanceOptions l_updateInstanceOptions ]
     [ ?updateDesignOptions l_updateDesignOptions ]
     [ ?updateSelectedNets l_updateSelectedNets ]
     [ ?updateSpecifiedLayers l_updateSpecifiedLayers ]
     [ ?updateUnattachedLabels l_updateUnattachedLabels ]
     [ ?loadPins s_updatePins ]
     [ ?mapperFile t_fileName ]
     [ ?addGeometries g_addGeometries ]
     [ ?addGeometryOptions l_addGeometryOptions ]
     [ ?updateBoundaries g_updateBoundaries ]
     [ ?updateBoundaryOptions l_updateBoundaryOptions ]
     [ ?replaceRows g_replaceRows ]
     [ ?replaceRowOptions l_replaceRowOptions ]
     [ ?replaceBlockages g_replaceBlockages ]
     [ ?replaceHalos g_replaceHalos ]
     [ ?replaceObstructionOptions l_replaceObstructionOptions ]
     [ ?transferConstraints g_transferConstraints ]
     )
     => t / nil
```

Description

Updates the instances, pins, geometries, and P&R information from a source cellview to a target cellview. The source cellview acts as a template to perform the following in the target cellview:

- Update instance location, orientation, and placement status.
- Update pin location, layer, height, and width.
- Add geometries such as shapes and wires.
- Replace P&R objects such as boundaries, rows, blockages, and halos.

Arguments

t_sourceLibName	Specify the name of the source library.

t_sourceCellName Specify the name of the source cell.

t_sourceViewName Specify the name of the source view.

t_targetLibName Specify the name of the target library.

t_targetCellName Specify the name of the target cell.

t_targetViewName Specify the name of the target view.

?updateInstances *g_updateInstances*

Specify if the instances are to be updated. The default value is `nil`.

?updateInstanceOptions *l_updateInstanceOptions*

Specify the instances to update in the target cellview. Set one or more of the following options to `t` or `nil`.

- `stdCells`
- `customCells`
- `macros`
- `ioPads`
- `addPhysOnlyInst`(instances that exist only in the physical domain and lack a corresponding instance in the logical domain. For example, filler cells)

?updateDesignOptions *l_updateDesignOptions*

Specify the parameters to be updated in the target cellview. Set one or more of the following options to `t` or `nil`.

- `cellType`
- `blockType`
- `symmetry` (orientation)
- `sitePattern`
- `trackPattern`

?updateSelectedNets *l_updateSelectedNets*

(Layout EXL Only) Specify the pins, shapes, and wires in the selected nets in the target cellview to be transferred from the source cellview. The default value is `nil`.

`?updateSpecifiedLayers` *`l_updateSpecifiedLayers`*

(Layout EXL Only) Copies pins, wires, and shapes from the source to the target cellview only for the specified target layers. The default value is `nil`.

`?updateUnattachedLabels` *`l_updateUnattachedLabels`*

(Layout EXL Only) Copies the unattached labels present in the source cellview to the target cellview. The default value is `t`.

`?loadPins` *`s_updatePins`* Specify if the pins are to be updated or replaced. Specify *`replace`* to load pins in the replace mode, *`update`* to load pins in the update mode, and *`nil`* to not load the pins. The default value is *`nil`*.

`?mapperFile` *`t_fileName`* (Layout EXL Only) Specify the name of a mapping file that defines the instance, layer, and layer purpose pair mapping between the source and target cellviews.

`?addGeometries` *`g_addGeometries`*

Specify if the geometries are to be added. The default value is `nil`.

`?addGeometryOptions` *`l_addGeometryOptions`*

Specify the geometries to be added from the source cellview to the target cellview at exactly the same locations as in the source cellview. The options are:

- `deleteGeometries`: Deletes existing shapes and wires (except Modgens and pins) before importing them from the source cellview.
- `addShapes`: Imports all the shapes from the source cellview to the target cellview.
- `addWiresAndVias`: Imports all the wires (pathSegs and vias) and their topologies from the source cellview to the target cellview.

`?updateBoundaries` *`g_updateBoundaries`*

Specify if the boundaries are to be updated. The default value is `nil`.

`?updateBoundaryOptions` *`l_updateBoundaryOptions`*

Specify the type of boundary for which information must be imported from the source cellview to the target cellview. The valid options are:

- `updatePRBoundary`
- `updateSnapBoundary`

`?replaceRows` *`g_replaceRows`*

Specify if the rows are to be replaced. The default value is `nil`.

`?replaceRowOptions` *`l_replaceRowOptions`*

Specify if the following types of rows are to be replaced.

- `replaceStandardRows`
- `replaceCustomRows`

`?replaceBlockages` *`g_replaceBlockages`*

Specify if the blockages are to be replaced. The default value is `nil`.

`?replaceHalos` *`g_replaceHalos`*

Specify if the halos are to be replaced. The default value is `nil`.

`?replaceObstructionOptions` *`l_replaceObstructionOptions`*

Specify the types of obstructions to be replaced in the target cellview with appropriate obstructions from the source cellview. Set one or more of the following options to `t` or `nil`.

- `placement`
- `routing`
- `slot`
- `pin`
- `fill`
- `feedThru`
- `screen`

`?transferConstraints` *`g_transferConstraints`*

- When set to `replace`, the constraints in the target cellview are replaced with the constraints transferred from the source cellview.
- When set to `update`, the constraints in the target cellview are updated.
- When set to `nil`, no constraints are transferred. The default value is `nil`.

Value Returned

`t`	Returns `t` if the function successfully updates the target cellview.
`nil`	Returns `nil` otherwise.

Examples

Updates information from the specified source cellview to the specified target cellview. The instances in the target cellview are updated. The pins in the target cellview are replaced with the ones in the source cellview. Geometries (shapes, wires, and vias) are added from the source to the target cellview.

```
vfpLoadPhysicalView("design" "top_level" "layout_xy" "design" "top_level" "layout"
?updateInstances t ?loadPins replace ?addGeometries t)
```

Updates information from the specified source cellview to the specified target cellview. Only the standard cells in the target cellview are updated. The pins in the target cellview are replaced with the ones in the source cellview. Geometries (shapes, wires, and vias) and boundaries are added from the source to the target cellview. Blockages and halos of the type placement are replaced in the target cellview with the ones in the source cellview.

```
vfpLoadPhysicalView("design" "top_level" "layout_xy" "design" "top_level" "layout"
?updateInstances t ?updateInstanceOptions (("stdCells" t)) ?loadPins replace
?addGeometries t ?updateBoundaries t ?replaceBlockages t ?replaceHalos t
?replaceObstructionOptions ("placement" t))
```

vfpPinConnectivitySetting

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

Description

This function takes no arguments. It invokes a GUI to enable the pin connectivity information setting.

Arguments

None

Value Returned

`t`	Returns `t` if the function successfully invokes the GUI
`nil`	Returns `nil` otherwise with an appropriate warning message

vfpPinSnapToEdge

```
vfpPinSnapToEdge(
     ?cv d_cv
     [ ?pinIOType { input | output | inputOutput } ]
     [ ?pinTermNames l_pinTermNames ]
     [ ?edge { Left | Right | Top | Bottom | Nearest } ]
     [ ?layerName t_layerName ]
     [ ?skipTracks x_numTracks ]
     [ ?honorCurrentLocation { t | nil } ]
     [ ?verboseMode { t | nil } ]
     )
     => t / nil
```

Description

Snaps pins to the specified edge. You can select pins by specifying either their I/O types or terminal names. You can also change the pin layers and skip tracks between pins.

Arguments

`?cv` *`d_cv`*

Specifies the cellview ID in which pins are to be snapped.

The default value is `nil`, which means that the current cellview is considered.

`?pinIOType { input | output | inputOutput }`

Specifies the I/O type of the pins to be snapped. The valid values are: "`input`", "`output`", and "`inputOutput`".

The default value is `""`.

`?pinTermNames` *`l_pinTermNames`*

Specifies the terminal names of the pins to be snapped, for example, '("`A`" "`B`").

To snap all the pins, set `pinTermNames` to `'("all")`.

The '.*' wildcard character and other regular expression constructs are supported to specify pin names. For example, to snap all the pins of a bus `A<0:9>`, set `pinTermNames = '(A.*)`.

Use either `pinIOType` or `pinTermNames` to specify the pins to be snapped. If both values are specified, then no snapping occurs.

`?edge { Left | Right | Top | Bottom | Nearest }`

Specifies the edge to which pins are to be snapped.

The valid values are "`Left`", "`Right`", "`Top`", "`Bottom`", and "`Nearest`". The default value is "`Nearest`".

`?layerName` *`t_layerName`*

Specifies the layer to which pins should be moved before snapping. For example, "`Metal1`" or "`Metal2`".

The default value is `""`, which means that pins remain on their current layers.

`?skipTracks` *`x_numTracks`*

Specifies the number of tracks to be skipped between the pins during snapping. For example, if `skipTracks=2`, then two tracks are skipped between the pins.

The default value is `0`.

`?honorCurrentLocation { t | nil }`

Honors the current pin location after pin distribution on the assigned edge. The valid values are `t` and `nil`. When set to `nil`, the optimal pin locations are based on the net length.

The default value is `t`.

`?verboseMode { t | nil }`

Prints additional information such as net lengths, error messages, and warnings while snapping pins. The valid values are `t` and `nil`. The default value is `nil`.

Value Returned

`t`	Pins were snapped to the specified edge.
`nil`	Pins were not snapped.

Examples

The following examples show how `vfpPinSnapToEdge` can be used with different argument combinations.

```
pinTermNamesList = '("all")
```

```
vfpPinSnapToEdge(
?cv geGetEditCellView()
?pinTermNames pinTermNamesList
)
```

In the above example, all the pins are snapped to the nearest edge.

```
pinTermNamesList = '("all")
vfpPinSnapToEdge(
?cv geGetEditCellView()
?pinTermNames pinTermNamesList
?edge "left"
)
```

In the above example, all the pins are snapped to the left edge.

```
pinTermNamesList = '("A" "B")
vfpPinSnapToEdge(
?cv geGetEditCellView()
?pinTermNames pinTermNamesList
?edge "left"
?layerName "Metal4")
)
```

In the above example, the pins with terminal names "`A`" and "`B`" are snapped to the left edge and the pin layers are changed to `Metal 4`.

```
vfpPinSnapToEdge
(
?cv geGetEditCellView()
?pinIOType "input"
?edge "left"
?layerName "Metal4"
)
```

In the above example, the pins with I/O type "`input`" are snapped to the left edge and the pin layers are changed to `Metal 4`.

```
pinTermNamesList = '("A" "B")
vfpPinSnapToEdge(
?cv geGetEditCellView()
?pinTermNames pinTermNamesList
```

```
?edge "left"
?skipTracks 2
)
```

In the above example, the pins with terminal names "A" and "B" are snapped to the left edge and two WSP tracks are skipped between the pins.

```
vfpPinSnapToEdge(
?cv geGetEditCellView()
?pinIOType "input"
?edge "left"
?honorCurrentLocation nil
)
```

In the above example, the pins with I/O type "input" are snapped to the left edge, and the pins snapped to the optimal track do not honor the current pin location.

vfpPtAddPinResizeEstimator

```
vfpPtAddPinResizeEstimator(
     t_functionName
     t_nickname
     )
     => t / nil
```

Description

Register a custom SKILL function for resizing pins. This registered function is passed as a parameter to `vfpPtRunPinResizeEstimator` to resize pins.

Arguments

t_functionName	Specifies the name by which the new SKILL function must be registered.
t_nickname	Specifies a nickname for the registered function. This is a simpler name that can be used to refer to the registered function.

Value Returned

`t`	The SKILL function has been registered.
`nil`	The SKILL function could not be registered.

Example

Registers a custom SKILL function:

```
vfpPtAddPinResizeEstimator("resizeFunction1" "nickName1")
```

vfpPtGetPinResizeEstimators

```
vfpPtGetPinResizeEstimators(
     [ g_nicknameOnly ]
     )
     => ( l_nickname l_functionName ) / nil
```

Description

Retrieves a list of registered pin resize estimator SKILL functions. You can specify whether the list must include only the function nicknames or both the function names and their nicknames.

Arguments

g_nicknameOnly	Specifies that only the nicknames of the registered pin resize estimator SKILL functions are to be listed. When set to `nil`, both the registered function names and their nicknames are displayed.

Value Returned

`( l_nickname l_functionName )`	List of names of registered pin resize estimator SKILL functions and their corresponding nicknames.
`nil`	There are no registered SKILL functions.

Example

Retrieves the function names and nicknames of registered pin resize estimator SKILL functions:

```
vfpPtGetPinResizeEstimators()
=>( ("nickname1" "functionName1" )  ("nickname2" "functionName2")...)
```

vfpPtLoadPinResizeFile

```
vfpPtLoadPinResizeFile(
     t_pathID
     )
     => t / nil
```

Description

Loads the specified SKILL file containing pin-resize functions.

Arguments

t_pathID	Path to the SKILL file to be loaded.

Value Returned

t	The file was successfully loaded.
nil	The file could not be loaded. An appropriate warning message is displayed.

Example

Loads the SKILL file containing pin resize functions.

```
vfpPtLoadPinResizeFile("/share/myFunc.il")
```

vfpPtRemovePinResizeEstimator

```
vfpPtRemovePinResizeEstimator(
     t_nickname
     )
     => t / nil
```

Description

Deregisters the given pin resize estimator SKILL function.

Arguments

t_nickname	Specifies the nickname of the pin resize estimator SKILL function that must be deregistered.

Value Returned

`t`	The pin resize estimator SKILL function was deregistered.
`nil`	The pin resize estimator SKILL function could not be deregistered.

Example

Deregisters the given pin resize estimator SKILL function:

```
vfpPtRemovePinResizeEstimator("nickname2")
```

vfpPtRunPinResizeEstimator

```
vfpPtRunPinResizeEstimator(
     d_topCellViewID
     t_nickname
     ?mode { all | selected | specify }
     ?pinList l_pinList
     )
     => t / nil
```

Description

Runs the given registered pin resize estimator SKILL function on pins as specified in the mode argument.

Arguments

d_topCellViewID

Identifies the top design cellview that contains the pins on which the function must be run.

t_nickname

Specifies the nickname of the pin resize estimator SKILL function to be run.

`?mode { all | selected | specify }`

Specifies the mode for running the SKILL function. The valid values are:

- `all`: The function is run on all top-level and level-1 pins.
- `selected`: The function is run on the selected pins.
- `specify`: The function is run on the pins that are specified in *ls_pinList*.

`?pinList` *ls_pinList*

Specifies a list of strings and regular expressions (separated by space and enclosed in double quotes) of the pins on which the function must be run.

Example: `"pin1 pin2 I1/p1 pa.* I1/p.*"`

Value Returned

`t`	The pin resize estimator SKILL function was run.
`nil`	The pin resize estimator SKILL function could not be run.

Example

Runs the given pin resize estimator SKILL function:

```
vfpPtRunPinREsizeEstimator(topCellView "nickname1" ?mode "specify" ?pinList "pin1
pinA* I1/pa*")
```

vfpPushPreRoutesDown

```
vfpPushPreRoutesDown(
     t_topCVLibName
     t_topCVCellName
     t_topCVViewName
     t_softBlockInstName
     t_targetLibName
     t_targetCellName
     t_targetViewName
     g_pushCloseWireBlkg
     g_rebindInst
     [ g_pushOverlappingBlockages ]
     [ g_pushPowerNetGeometry ]
     [ g_pushRegularNetGeometry ]
     [ g_pushRow ]
     [ g_pushWSP ]
     [ g_pushWholeShape ]
     [ g_pushSelection ]
     [ g_pushFloatingShapes ]
     [ t_pushMode ( Auto Push | Create Net & Push with Feed Thru Pins | Push
     as Blockage ) ]
     [ l_pushLPPs ]
     )
     => t / nil
```

Description

Pushes the top-level power structures of a cellview over a specified soft block instance into a target cellview.

Arguments

t_topCVLibName	Specifies the library name of the top cellview
t_topCVCellName	Specifies the cell name of the top cellview
t_topCVViewName	Specifies the view name of the top cellview
t_softBlockInstName	Specifies the soft block instance name of the top cellview
t_targetLibName	Specifies the target library name
t_targetCellName	Specifies the target cell name
t_targetViewName	Specifies the target view name

g_pushCloseWireBlkg Specifies if the wires that are outside but close to the PR boundary are to be pushed as blockage. Specify `t` to push as blockage, `nil` otherwise. Default value is `nil`.

g_rebindInst Specifies if the instance are to be remastered with the target view. Specify `t` to remaster, `nil` otherwise. This is an optional argument. Default value is `nil`.

g_pushOverlappingBlockages

Species whether or not to push blockages into instances. Default value is `nil`.

g_pushPowerNetGeometry

Specifies whether power nets must be selected. Default value is `t`.

g_pushRegularNetGeometry

Specifies whether nets apart from power/ground nets must be selected. Default value is `nil`.

g_pushRow Specifies whether standard and custom rows must be pushed inside blocks. Default value is `nil`.

g_pushWSP Specifies whether top-level WSP patterns must be pushed into sub-blocks such that the WSP pattern at the top-level and the WSP pattern inside the soft block are aligned. Default value is `nil`.

g_pushWholeShape Specifies whether the whole shapes that overlap the selected soft block are to be pushed as they are or must be clipped when they are pushed inside blocks.

g_pushSelection Specifies whether only the selected nets and shapes are to be pushed inside the soft block.

g_pushFloatingShapes Selects all floating net shapes, which are net shapes without any connections.

t_pushMode (Auto Push | Create Net & Push with Feed Thru Pins | Push as Blockage)

	Specifies one of the following push modes for shapes: ■ `Auto Push`: Automatically pushes all shapes into the selected soft blocks. Unconnected shapes are pushed in as blockages. ■ `Create Net & Push with Feed Thru Pins`: Creates missing nets before pushing all shapes into the selected soft blocks. Unconnected shapes are pushed in as feed-thru terminals. ■ `Push as Blockage`: Pushes all shapes as blockages.
l_pushLPPs	Specifies the layers that contain the shapes that are to be pushed into soft blocks.

Value Returned

`t`	The top-level power-structures of the given cellview were pushed into the specified soft block.
`nil`	The command was unsuccessful.

Example

Pushes the top-level power-structures of the given cellview into the specified soft block.

```
vfpPushPreRoutesDown("design" "top_demo" "layout_cases" "I4" "design" "block2"
"layout_pushed_skill" t t)
```

vfpReportIOPadLocation

```
vfpReportIOPadLocation(
     d_dbCellviewID
     g_printFillerPads
     )
     => t / nil
```

Description

Prints the IO pad information in formatted manner for all the pads which are placed in the IO rows within the design.

Arguments

d_dbCellviewID	Specifies the design cellview ID for which the IO report is required.
g_printFillerPads	Prints information about the IO filler pads present in the IO rows along with other IO pad information. The information is printed only when this value is set to `t`.

Value Returned

`t`	The function successfully generated the report in the CIW.
`nil`	Returns `nil` with an appropriate warning message if the report was not generated.

Examples

To print only pad locations, specify the following:

```
vfpReportIOPadLocation (cellviewID)
```

This does not print filler pad instances that are present in the design.

To print filler pad instances as well, specify the following:

```
vfpReportIOPadLocation (cellviewID t)
```

vfpSbDefineObstruction

```
vfpSbDefineObstruction(
     l_PhysLCV
     t_obstructionType
     [ ?topOffset  n_topOffset ]
     [ ?bottomOffset n_BottomOffset ]
     [ ?leftOffset n_LeftOffset ]
     [ ?rightOffset n_RightOffset ]
     [ ?layerName t_layerName ]
     )
     => t / nil
```

Description

Creates placement, routing, or other obstructions based on the `obstructionType` argument. All conditions that are enforced for these obstructions through GUI-based flow are applicable here. For example, only one placement blockage would be applied per soft block. If a second placement blockage is applied through this API, a warning is issued. The latest applied placement blockage will not replace the existing placement blockage.

Arguments

l_PhysLCV	List of ("lib" "cell" "view") triplets
t_obstructionType	Type of obstruction to be added. The valid values are: "placement", "routing", "fill", "slot", "pin", "feedthru", and "screen".
?topOffset *n_topOffset*	Top offset of the obstruction
?bottomOffset *n_BottomOffset*	Bottom offset of the obstruction
?leftOffset *n_LeftOffset*	Left offset of the obstruction
?rightOffset *n_RightOffset*	Right offset of the obstruction
?layerName *t_layerName*	Layer name. This argument is ignored when the `obstructionType` is “placement”.

Value Returned

`t`	The obstruction is created.
`nil`	The command was unsuccessful.

Example

```
vfpSbDefineObstruction(list("design" "block1" "layout") "fill" ?leftOffset 20
?rightOffset 30 ?bottomOffset 50 ?topOffset 2 ?layerName "metal2")
```

vfpSbDeleteObstruction

```
vfpSbDeleteObstruction(
     l_PhysLCV
     t_obstructionType
     [ ?topOffset  n_topOffset ]
     [ ?bottomOffset n_BottomOffset ]
     [ ?leftOffset n_LeftOffset ]
     [ ?rightOffset  n_RightOffset ]
     [ ?layerName t_layerName ]
     )
     => t / nil
```

Description

Deletes the specified obstruction.

Arguments

l_PhysLCV	List of ("lib" "cell" "view") triplets
t_obstructionType	Type of obstruction to be deleted. The valid values are: "placement", "routing", "fill", "slot", "pin", "feedthru", and "screen".
?topOffset *n_topOffset*	Top offset of the obstruction
?bottomOffset *n_BottomOffset*	Bottom offset of the obstruction
?leftOffset *n_LeftOffset*	Left offset of the obstruction
?rightOffset *n_RightOffset*	Right offset of the obstruction
?layerName *t_layerName*	Layer name. This argument is ignored when the `obstructionType` is “placement”.

Value Returned

`t`	The obstruction was successfully deleted.
`nil`	The command was unsuccessful.

Example

```
vfpSbDeleteObstruction(list("design" "block1" "layout") "placement" ?leftOffset 20
?rightOffset 30 ?bottomOffset 50 ?topOffset 2 ?layerName "metal1")
```

vfpSbEditIOPin

```
vfpSbEditIOPin(
     l_PhysLCV
     t_termName
     [ ?lpp txl_lpp ]
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?number x_number ]
     [ ?criticality x_criticality ]
     [ ?sigType t_sigType ]
     )
     => t / nil
```

Description

Edits attributes of the pin that is associated with the specified `termName`.

Arguments

l_PhysLCV	List of ("lib" "cell" "view") triplets
t_termName	Terminal name corresponding to the pin
?lpp *txl_lpp*	Layer Purpose Pair. The valid formats are text, number, and list. Use one of the following arguments to identify the layer purpose pair. **Text:** "layerName" – The default purpose is "drawing" “layerName purposeName” **Number:** layerNum – The default purpose is "drawing" **List:** list (“layerName” “purposeName”) list (layerNum, ”purposeName”)
?width *n_width*	Width of the Pin
?height *n_height*	Height of the Pin
?number *x_number*	Number of `pinFigs` to be created on the pin. The maximum number of `pinFigs` that can be created is 20.

`?criticality` *`x_criticality`*

Criticality of the terminal (-128 to 128)

`?sigType` *`t_sigType`* Signal type of the terminal

The valid signal types are: "`signal`", "`ground`", "`supply`", "`clock`", "`analog`", "`tieOff`", "`tieHi`", "`tieLo`", "`scan`", and "`reset`".

Value Returned

`t`	The pin attributes were changed.
`nil`	The command was unsuccessful.

Examples

```
vfpSbEditIOPin(list("design" "block1" "layout") "in1" ?lpp "metal2")

vfpSbEditIOPin(list("design" "block1" "layout") "in1" ?lpp 20 ?width 10.0)

vfpSbEditIOPin(list("design" "block1" "layout") "in1" ?lpp list("metal3" "pin")
?criticality 20)
```

vfpSbEditSoftBlockType

```
vfpSbEditSoftBlockType(
     l_PhysLCV
     t_blockType
     )
     => t / nil
```

Description

Edits the blockType of the specified soft block.

Arguments

l_PhysLCV	List of ("lib" "cell" "view") triplets
t_blockType	The block type. The valid values are "CUSTOM" and "DIGITAL". This argument is case-insensitive.

Value Returned

t	The block type was changed.
nil	The command was unsuccessful.

Example

```
vfpSbEditSoftBlockType(list("design" "block1" "layout") "custom")
```

vfpSbSetPolygonalBoundary

```
vfpSbSetPolygonalBoundary(
     l_PhysLCV
     l_ listOfPolygonPoints
     )
     => t / nil
```

Description

Specifies the boundary attributes of soft blocks that are polygonal in shape.

Arguments

l_PhysLCV	List of ("lib" "cell" "view") triplets
l_listOfPolygonPoints	The vertices of the rectilinear polygon in the form of a list of x, y coordinates

Value Returned

`t`	The vertices of the polygonal soft block were set.
`nil`	The vertices of the polygonal soft block were not set.

Example

```
vfpSbSetPolygonalBoundary(list("design" "top_viewtest" "layout_sav") list('(0 0)
'(0 500000) '(500000 500000) '(500000 0)))
```

vfpSbSetRectangularBoundary

```
vfpSbSetRectangularBoundary(
     l_PhysLCV
     n_width
     n_height
     )
     => t  / nil
```

Description

Specifies the boundary vertices of the rectangular soft block.

Arguments

l_PhysLCV	List of ("lib" "cell" "view") triplets
n_width	Width of the rectangular boundary
n_height	Height of the rectangular boundary

Value Returned

t	The boundary vertices were set.
nil	The command was unsuccessful.

Example

```
vfpSbSetRectangularBoundary(list("design" "block1" "layout") 20 30)
```

vpaOptimizePins

```
vpaOptimizePins(
     ?cv d_cellViewId
     ?layoutLCV l_LCVName
     [ ?cphId g_cphId ]
     [ ?selectedOnly g_selectedOnly ]
     [ ?spacingType t_spacingType ]
     [ ?spacingValue n_spacingValue ]
     [ ?targetPinSide l_targetPinSides ]
     [ ?enableCongestion g_enableCongestion ]
     [ ?congestionValue n_congestionValue ]
     )
     => t / nil / List of (sbId lib:cell:view status)
```

Description

Runs pin optimization on the specified cellview or design based on the options you specify. You can specify only those parameters that you want to control. The parameters that you do not specify get their values from the corresponding LayoutXL environment variable.

Arguments

`?cv` *d_cellViewId*	The ID of the cellview for pin optimization. This cellview ID is mutually exclusive with *layoutLCV* and has a higher priority than *layoutLCV*.
`?layoutLCV` *l_LCVName*	The name of the library, cell, and view. It is the target cellview for Pin optimization and is mutually exclusive with *cv*. *Important* Either *cv* or *layoutLCV* must be specified.
`?cphId` *g_cphId*	The physical configuration ID, which is the physical configuration cellview created by the `cphCreatePhysConfig` API. The `cphId` must only be specified when using the pin optimizer in conjunction with the command-line CPH-GPH-based flow. If this argument is not used, then the pin optimizer retains its original behavior without any impact of the command-line CPH-GPH flow.
`?selectedOnly` *g_selectedOnly*	

Whether to run pin optimization on the selected objects only or all the objects in cellview. The valid values are `t` and `nil`. The default value is read from the corresponding LayoutXL Floorplan environment variable.

`?spacingType` *`t_spacingType`*

Specifies how to space out the pins using the relevant spacing values from the technology file. The valid values are `line-line`, `line-via`, `via-via`, or `specify`. The default value is set to `line-line` in the corresponding environment variable . If the value is set as `specify`, then the *spacingValue* has to be specified.

`?spacingValue` *`n_spacingValue`*

This value needs to be specified if the *spacingType* is set to `specify`. This value is used to specify the minimum spacing between the pins. The value is either a fixed or a floating number.

`?targetPinSide` *`l_targetPinSides`*

The side(s) of the blocks and the PR boundary on which the pins should be placed. The valid values are `left`, `right`, `top`, and `bottom`. By default all the four sides are considered. You can also specify a combination of one or more sides.

`?enableCongestion` *`g_enableCongestion`*

Whether or not to consider the congestion value during pin optimization. This option is only applicable if congestion data is available. The valid values are `t` or `nil`. If the value is set to `t`, then the *congestionValue* has to be specified. The default value is read from the corresponding LayoutXL Floorplan environment variable.

`?congestionValue` *`n_congestionValue`*

Specifies the congestion threshold. This needs to be specified if *enableCongestion* is `t`. It is a fixed number.

Value Returned

When a valid *cphId* is not specified, one of the following values is returned:

`t`	Pin optimization was successfully run. The Layout XL Floorplan environment variables were updated as per the specified values.
No value returned	Pin optimization failed on some of the blocks (partially) or on all blocks (completely). Related error messages are displayed in the CIW.

When a valid *cphId* is specified, one of the following values is returned:

`t`	The Pin Optimizer was successfully run.
`nil`	The Pin Optimizer completely failed on all blocks.
List of (*sbId lib:cell:view status*)	The Pin Optimizer partially failed on some blocks. A list of soft block IDs, their *lib:cell:views*, and their statuses is returned. The list of blocks indicates whether pin optimization was performed for each soft block.

```
(
(sbId  lib:cell:view  status)
(sbId lib:cell:view   status)
)
```

- `sbId` – ID of the soft-block that was created using `cphSbDefineSoftBlock` API
- `lib:cell:view` – Physical LCV of the soft block
- `status` – Status of the soft block after running Pin Optimizer. Values returned are:
  - `t` – Soft block is successfully optimized.
  - `nil` – Soft block could not be optimized.

Examples

To optimize only the selected pins in the cellview `design:top:layout` by placing them on either of the four edges with `0.1` separation.

```
vpaOptimizePins(?layoutLCV '("design" "top" "layout") ?selectedOnly t ?spacingType
"specify" ?spacingValue 0.1)
```

To optimize all the pins in the cellview with ID `d_cvId` by placing them on the `left` and `right` edges with `line-line` separation.

```
vpaOptimizePins(?cv d_cvId ?cphID mycphID ?spacingType "line-line" ?targetPinSide
("left" "right"))
```

8

Module Generator Functions

This topic lists the Cadence® SKILL functions associated with the Virtuoso Module Generator (Modgen).

Only the functions listed below are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Modgen Editor Functions

- mgAddBodyContact
- mgAbutCB
- mgAddBodyContact
- mgAddCopyDummies
- mgAddDummyBottomCB
- mgAddDummyLeftCB
- mgAddDummyRCBottomCB
- mgAddDummyRCLeftCB
- mgAddDummyRCRightCB
- mgAddDummyRCTopCB
- mgAddDummyRightCB
- mgAddDummyTopCB
- mgAddEmptyRCCB
- mgAddGuardRingCB
- mgAddShapeToTopo

- mgAddTopologyToModgen
- mgCreateMatchGroupInModgen
- mgCreateModgenAsLayout
- mgCreateModgenConstraintAsLayout
- mgCreateOrEdit
- mgClearDummyLibCellAndParamAlias
- mgDeleteAllBodyContacts
- mgDeleteAllDummies
- mgDeleteAllDummyRowColumnCB
- mgDeleteAllEmptyRowColumnCB
- mgDeleteCB
- mgDeleteEmptyRowColumnCB
- mgDestroyMatchGroupOnObject
- mgEditGuardRingCB
- mgExecNoConObs
- mgExitCB
- mgEvalInBackgroundModgen
- mgFGREnterHandEdit
- mgFGRExitHandEdit
- mgFGRIsHandEdit
- mgFlattenModgens
- mgFlipHorizontalCB
- mgFlipVerticalCB
- mgGetBoxLPPArea
- mgGetChannelTrunks
- mgGetColumnRoutingChannelWidth
- mgGetConstraintFromFG

-
- mgGetDummyRefDirectionParams
- mgGetGuardRingConnectObject
- mgGetIsLocalTrunk
- mgGetIsLocalTrunk
- mgGetLayerSpacing
- mgGetLayerSpacing
- mgGetMatchGroupMembers
- mgGetModgenConstraintFromTopology
- mgGetModgenFigGroupFromTopology
- mgGetModgenFGFromConstraint
- mgGetRegisteredDummyNetProc
- mgGetRegisteredDummyOverrideParameters
- mgGetRegUserProc
- mgGetRowRoutingChannelWidth
- mgGetStrapDirection
- mgGetStrapHasDirection
- mgGetStrapHasOffset
- mgGetStrapOffset
- mgGetStrapLongOffset1
- mgGetStrapLongOffset2
- mgGetTopologyFromModgen
- mgGetTopoTrunkShapes
- mgGetTopoShapes
- mgGetTrunkChannel
- mgGetTrunkRefLayerPurpose
- mgGetTrunkRefLPPEnclosure

- mgGetTrunkTopo
- mgHilightEmptyRowColumnCB
- mgIsInBackAnnotation
- mgIsTopologyInsideModgen
- mgModgenHasTopology
- mgObjectHasMatchGroup
- mgRegenerateModgen
- mgRegisterDummyNetProc
- mgRegisterDummyOverrideParameters
- mgRegisterDummyParamsRefDirection
- mgRegUserProc
- mgRemoveMemberFromMatchGroup
- mgRemoveTopologyFromModgen
- mgRotateLeftCB
- mgRotateMXCB
- mgRotateMYCB
- mgRotateR270CB
- mgRotateR90CB
- mgRotateRightCB
- mgRouteCB
- mgRoutePToTCB
- mgSelectRowColCB
- mgSelectRowColCB
-
- mgSetIsLocalTrunk
- mgSetStrapDirection
- mgSetStrapHasDirection

- mgSetStrapHasOffset
- mgSetStrapLongOffset1
- mgSetStrapLongOffset2
- mgSetStrapOffset
- mgSetTrunkRefLayerPurpose
- mgSetColumnRoutingChannelWidth
- mgSetRowRoutingChannelWidth
- mgSwapCB
- mgUnAbutCB
- gpeRegisterPresetGen
- mgUnregisterDummyOverrideParameters
- mgUnregisterDummyParamRefDirection
- mgUnRegUserProc
- mgUpdateCB
- mgUpdateCB
- mgUpdateEmptyRowColumnHilightCB
- mgUpdateHoriAlignCB
- mgUpdateVertAlignCB

Modgen Pattern Editor Functions

- gpeAddInstance
- gpeClearPresetGenerators
- gpeExtractReuseTemplate
- gpeExtractReuseTemplatesFromLCV
- gpeExtractTemplateFromMG
- gpeFindArrayPatternDiffs
- gpeGetGridValue

- gpeGetSelection
- gpeIsPresetGenDisplayable
- gpeIsRegisteredPresetGen
- gpeLoadReuseTemplate
- gpeMove
- gpePresetReorientResistor
- gpeRegisterPresetGen
- gpeRemoveInstance
- gpeRunPresetGen
- gpeSelect
- gpeSetGridText
- gpeSetGridValue
- gpeSetOrientText
- gpeSetSize
- gpeUnregisterPresetGen
- gpeUpdateInstanceFromSchematic

Modgen Placement Functions

Modgen Placement and Routing Functions

- gpeAbutGridEntries
- gpeAddDummy
- gpeAddDummyInEmptyCells
- gpeAddDummySurround
- gpeCancelSandbox
- gpeClearGridEntries
- gpeCopyColAbutment
- gpeCopyRowAbutment

- gpeCreateAbutEntries
- gpeCreateAbutType
- gpeCreateAlignment
- gpeCreateAlignmentAndSpacing
- gpeCreateDummyConfig
- gpeCreateDummyConfigNet
- gpeCreateDummyConfigParam
- gpeCreateGridEntry
- gpeCreateMapEntry
- gpeCreateSandbox
- gpeDeleteDummyEntries
- gpeDeleteSandbox
- gpeEditSandbox
- gpeFinishSandbox
- gpeGetAbut
- gpeGetAbutTypeList
- gpeGetAbutTypeName
- gpeGetConstraints
- gpeGetDefaultAbutType
- gpeGetFigGroup
- gpeGetGrid
- gpeGetGridColCount
- gpeGetGridEntry
- gpeGetGridEntries
- gpeGetGridRowCount
- gpeGetGridSelection
- gpeGetMap

- gpeGetMapEntry
- gpeInsertEmptyColumns
- gpeInsertEmptyRows
- gpeIsSandboxSync
- gpeIsValidAbutType
- gpePlaceGridEntries
- gpeRemoveEmptyRowsAndColumns
- gpeSboxp
- gpeSetAbut
- gpeSetAbutEntries
- gpeSetGrid
- gpeSetGridEntry
- gpeSetMap
- gpeSetMapEntry
- gpeSetMergeLayer
- gpeStacksAreCompressed
- gpeStacksCompress
- gpeStacksUncompress
- gpeStartSandbox
- gpeSyncSandbox
- gpeUnAbutGridEntries
- gpeUnSyncSandbox

Modgen Routing Functions

- gpeAddMatchGroups
- gpeAddStrapEntries
- gpeAddTrunkChains

- gpeClearChannelWidthEntry
- gpeClearMatchGroups
- gpeClearStrapEntries
- gpeClearTopo
- gpeClearTrunkChains
- gpeCreateChannelWidthEntry
- gpeCreateInstTermEntries
- gpeCreateMatchGroupEntry
- gpeCreateStrapEntries
- gpeCreateTrunkChain
- gpeCreateTrunkEntries
- gpeCreateTwigEntries
- gpeGetChannelWidthEntry
- gpeGetStrapEntries
- gpeGetTrunkChains
- gpeGetTwigEntries
- gpeSetChannelWidthEntry
- gpeSetRouter

Deleted Module Generator Functions

The following Module Generator functions are no longer supported:

- mgGen
- mgRouteStructuredCB
- mgUnRouteCB
- mgPatternCB

Related Topics

Module Generator Functions

mgAbutAllCB

```
mgAbutAllCB(
     g_abutFlag
     )
     => t / nil
```

Description

Abuts all currently selected instances.

Arguments

g_abutFlag	This is an optional argument. If the value for this argument is 1, it indicates to abut all devices. However, if the value is 0, it means to unabut all devices. By default, the value is 1.

Value Returned

`t`	The selected instances are abutted.
`nil`	The selected instances are not abutted.

Example

Abuts all currently selected instances.

```
mgAbutAllCB(1)
```

mgAbutCB

```
mgAbutCB(
     d_cellViewID
     )
     => t / nil
```

Description

Attempts to abut the selected instances in the Modgen currently being edited.

Arguments

d_cellViewID	Database ID of the cellview.

Value Returned

t	The selected instances are successfully abutted.
nil	The selected instances are not abutted.

Example

Abuts the selected instances in the Modgen currently being modified in the specified cellview.

```
mgAbutCB(cv)
```

Related Topics

mgAddBodyContact

```
mgAddBodyContact(
     s_side
     )
     => t / nil
```

Description

Adds a body contact on the specified side to the selected instances inside a Modgen. Side can be one of *left*, *right*, *top*, or *bottom*.

Arguments

s_side	The side on which to add a body contact. Valid values are *left*, *right*, *top*, or *bottom*.

Value Returned

t	The body contact is added.
nil	The body contact is not added.

Example

Adds a body contact to the left side of any selected instances in the Modgen currently being modified.

```
mgAddBodyContact("left")
```

mgAddCopyDummies

```
mgAddCopyDummies(
     d_mgID
     t_sideString
     S_netName
     l_insts
     )
     => t / nil
```

Description

Creates copies of the specified Modgen dummy and places them on the *t_sideString* side of the instance.

Arguments

d_mgID	Modgen figGroupId for which dummies need to be created.
t_sideString	Side of the instance where the dummies need to be placed. Valid values are `left`, `right`, `top`, and `bottom`.
S_netName	Name of the net to which the dummies need to be connected.
l_insts	List of instances.

Value Returned

`t`	The dummy copies were created.
`nil`	The dummy copies were not created.

Example

Creates copies of the specified dummies on the left of the instance "`|M5`" and "`|M6`":

```
mgId = leGetEditFigGroup()
cv = geGetEditCellview()
inst1 = dbFindAnyInstByName(cv "|M5")
inst2 = dbFindAnyInstByName(cv "|M6")
mgAddCopyDummies(mgId "left" "vdda!" list(inst1 inst2))
```

mgAddDummyBottomCB

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

Description

Adds dummy devices to the bottom of the selected instances.

Arguments

None

Value Returned

`t`	The dummy devices are added.
`nil`	The dummy devices are not added.

Example

Adds a dummy to the bottom of the selected instances.

```
mgAddDummyBottomCB()
```

mgAddDummyLeftCB

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

Description

Adds dummy devices to the left side of the selected instances.

Arguments

None

Value Returned

`t`	The dummy devices are added.
`nil`	The dummy devices are not added.

Example

Adds a dummy to the left side of the selected instances.

```
mgAddDummyLeftCB()
```

mgAddDummyRCBottomCB

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

Description

Adds a complete row of dummy devices to the bottom of the selected instances.

Arguments

None

Value Returned

`t`	The row of dummy devices was added.
`nil`	The row of dummy devices was not added.

Example

Adds a row of dummies to the bottom of the selected instances.

```
mgAddDummyRCBottomCB()
```

mgAddDummyRCLeftCB

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

Description

Adds a complete row of dummy devices to the left of the selected instances.

Arguments

None

Value Returned

`t`	The row of dummy devices was added.
`nil`	The row of dummy devices was not added.

Example

Adds a row of dummies to the left of the selected instances.

```
mgAddDummyRCLeftCB()
```

mgAddDummyRCRightCB

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

Description

Adds a complete row of dummy devices to the right of the selected instances.

Arguments

None

Value Returned

`t`	The row of dummy devices was added.
`nil`	The row of dummy devices was not added.

Example

Adds a row of dummies to the right of the selected instances.

```
mgAddDummyRCRightCB()
```

mgAddDummyRCTopCB

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

Description

Adds a complete row of dummy devices to the top of the selected instances.

Arguments

None

Value Returned

`t`	The row of dummy devices was added.
`nil`	The row of dummy devices was not added.

Example

Adds a row of dummies to the top of the selected instances.

```
mgAddDummyRCTopCB()
```

mgAddDummyRightCB

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

Description

Adds dummy devices to the right side of the selected instances.

Arguments

None

Value Returned

`t`	The dummy devices are added.
`nil`	The dummy devices are not added.

Example

Adds a dummy to the right side of the selected instances.

```
mgAddDummyRightCB()
```

mgAddDummyTopCB

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

Description

Adds dummy devices to the top of the selected instances.

Arguments

None

Value Returned

`t`	The dummy devices are added.
`nil`	The dummy devices are not added.

Example

Adds a dummy to the top of the selected instances.

```
mgAddDummyTopCB()
```

mgAddEmptyRCCB

```
mgAddEmptyRCCB(
     t_side
     )
     => nil
```

Description

Adds an empty row or column to the specified side of a selected instance. To add empty row or column only one instance should be selected.

Argument

t_side	Specifies the string argument that can be `left`, `right`, `top`, or `bottom`.

Examples

In this example, an empty column is added to the left of the selected instance.

```
mgAddEmptyRCCB("left")
```

In this example, an empty row is added to the top of the selected instance.

```
mgAddEmptyRCCB("top")
```

mgAddGuardRingCB

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

Description

Displays the Guard Ring Options form for creating and modifying the guard ring for the current Modgen.

Arguments

None

Value Returned

`t`	The Guard Ring Options form is displayed.
`nil`	The Guard Ring Options form is not displayed.

Example

Displays the Guard Ring Options form for the current Modgen.

```
mgAddGuardRingCB()
```

mgAddShapeToTopo

```
mgAddShapeToTopo(
     d_topoId
     d_figId
     )
     => t / nil
```

Description

Adds a user-generated shape, such as a rectangle, to a topology group, usually a topological strap.

Arguments

d_topoId	Database ID of the topology object or strap to which the shape is to be added.
d_figId	Database ID of the shape to add.

Value Returned

t	The shape was added to the topology group.
nil	The command was unsuccessful.

Example

Adds a rectangle to the specified topological group.

```
cv = geGetEditCellView()
rect = dbCreateRect(cv "Metal2" list(10:10 12:10.2))
topos = dbGetCellViewTopologies(cv)
inpTopos = car(exists(topo topos dbGetTopologyNet(topo)~>name == "INP"))
strap = car(dbGetTopologyObjects(inpTopos))
mgA
ddShapeToTopo(strap, rect)
```

mgAddTopologyToModgen

```
mgAddTopologyToModgen(
     d_tpObjectId
     d_modgenId
     )
     => t / nil
```

Description

Adds the specified topology object to the specified Modgen.

Arguments

d_tpObjectId	A topology object ID
d_modgenId	A Modgen figGroup, storage group, or constraint ID

Value Returned

`t`	The topology object was added to the specified Modgen.
`nil`	The command was unsuccessful.

Example

Adds topology objects to Modgen with `mgId` and `cid` as the Modgen constraint ID:

```
cv = mgId->cellView
netA = dbFindNetByName(cv "net5")
netB = dbFindNetByName(cv "net7")
when(and(netA netB)
    topoA = dbCreateTopology("TopologyA" netA)
    topoB = dbCreateTopology("TopologyB" netB)
    when(and(topoA topoB)
    mgAddTopologyToModgen(topoA cid)
    mgAddTopologyToModgen(topoB cid)
    ) ; end-when
) ; end-when
```

mgCreateMatchGroupInModgen

```
mgCreateMatchGroupInModgen(
     d_modgenConId
     l_tpObjects
     t_twigMatchType
     [ t_trunkMatchType ]
     )
     => t / nil
```

Description

Creates a matchGroup in the specified Modgen storage.

Arguments

d_modgenConId	Modgen Constraint ID.
l_tpTrunkObjects	List of trunk topology objects.
t_twigMatchType	The match criteria for creating twigs that are connected to the trunks specified in the second argument. Value `unknownType` indicates that the twigs do match. Valid values are: ■ `lengthenOnly` ■ `widenOnly` ■ `lengthenFirst` ■ `widenFirst` ■ `unknownType`
t_trunkMatchType	The match criteria to be used to trim the trunks specified in the second argument. Valid values are: ■ `matchLength` - Trims trunks such that all the trunks in the list will have the same length. ■ `unknownType` (default) - Trunks do not match. If the argument is not specified, the default value is considered.

Value Returned

`t`	A matchGroup was created.
`nil`	Failed to create the matchGroup.

Example

Creates a matchGroup for `netA` with the `widenFirst` twig match type in the specified Modgen, with the Modgen figGroup ID set to `mgId` and the Modgen constraint ID set to `cid`. Also creates a matchGroup for `netB` with the `lengthenOnly` twig match type and trunk match type `matchLength`.

```
cv = mgId->cellView
netA = dbFindNetByName(cv "net5")
netB = dbFindNetByName(cv "net7")
when(and(netA netB)
  topoA = dbCreateTopology("TopologyA" netA)
  topoB = dbCreateTopology("TopologyB" netB)
  when(and(topoA topoB)
    trunkAA = dbCreateTrunk("TrunkAA" topoA "horizontal"))
    trunkAB = dbCreateTrunk("TrunkAB" topoA "horizontal"))
    trunkBA = dbCreateTrunk("TrunkBA" topoB "horizontal"))
    trunkBB = dbCreateTrunk("TrunkBB" topoB "horizontal"))
    when(and(trunkAA trunkAB trunkBA trunkBB)

      ;; Set the layers, widths, anchors and orthogonal offsets of the
      ;; trunks as appropriate here.


      mgCreateMatchGroupInModgen(cid list(trunkAA trunkAB) "widenFirst"
"unknownType")
      mgCreateMatchGroupInModgen(cid list(trunkBA trunkBB) "lengthenOnly"
"matchLength")
      mgAddTopologyToModgen(topoA cid)
      mgAddTopologyToModgen(topoB cid)
      mgAddTopologyToModgen(trunkAA cid)
      mgAddTopologyToModgen(trunkAB cid)
      mgAddTopologyToModgen(trunkBA cid)
      mgAddTopologyToModgen(trunkBB cid)
    ) ; end-when
  ) ; end-when
) ; end-when
```

mgCreateModgenAsLayout

```
mgCreateModgenAsLayout(
     d_cvId
     [ l_InstanceList ]
     )
     => figGroupID / nil
```

Description

Creates a Modgen from the list of specified instances and keeps the relative instance locations inside the Modgen, as similar as possible to the original instances in the layout. If instance list is not specified, the selected instances from the cellview are used.

Arguments

d_cvId	The layout cellView ID where the selected instances are located. If set to `nil`, then the current cellview is used.
l_InstanceList	List of instances.

Value Returned

figGroupID	The figGroup ID of the Modgen that was created.
`nil`	A Modgen could not be created.

Examples

Creates a Modgen in the current editing cellview (`geGetEditCellView`) that includes all instances current selected (`geGetSelSet`).

```
mgCreateModgenAsLayout(geGetEditCellView() geGetSelSet())
```

mgCreateModgenConstraintAsLayout

```
mgCreateModgenConstraintAsLayout(
     d_cvId
     l_InstanceList
     [ l_AbuttedInstances ]
     )
     => g_constraintID / nil
```

Description

Creates a Modgen constraint that uses the current layout to drive the initial Modgen constraint and row/column assignments.

Arguments

d_cvId	A valid layout cellview ID. If the value is invalid, the function returns `nil`.
l_InstanceList	A non-empty list of instance IDs. If the values are invalid, the function returns `nil`.
l_AbuttedInstances	A list of abutted instances for which abutment needs to be preserved while generating the Modgen from the layout. Example: Consider abutment M0 <- M1; M2<-M3 in the layout. Instance names M1 and M3 is passed as a list for the *l_AbuttedInstances* argument.

Value Returned

g_constraintID	Constraint ID of the Modgen that was created.
`nil`	A Modgen could not be created.

Examples

Creates a Modgen constraint that uses the specified instances.

```
window = hiGetCurrentWindow()
cvId = geGetEditCellView(window)
when(cvId~>objType == "cellView" && cvId~>cellViewType == "maskLayout"
    instances = cellView~>instances
    when(instances
    mgCid = mgCreateModgenConstraintAsLayout(cvId instances)
    )
)
```

The following example works only if the current cellview is of the layout type. `geGetEditCellView` returns a valid layout cellview ID. The Modgen includes the instances that are selected in the cellview.

```
mgCid = mgCreateModgenConstraintAsLayout(geGetEditCellView()  geGetSelSet())
```

mgCreateOrEdit

```
mgCreateOrEdit(
     d_cellViewId
     d_constraintId
     [ d_figGroupId ]
     )
     => t / nil
```

Description

Opens the Modgen editor to create a new Modgen or to edit parameters of an existing Modgen. After making edits, the Modgen constraint is updated. If the source is a layout cellview, then the Modgen layout, the associated geometry, and the associated database objects are updated.

Arguments

d_cellViewId	Database ID of the cellview. If set to `nil`, then the current cellview is used.
d_constraintId	Database ID of the constraint.
d_figGroupId	Database ID of the Modgen figGroup. If both, the constraint and figGroup are set to `nil`, then the current selection is used to find a Modgen to edit or instances to create a Modgen with.

Value Returned

`t`	The Modgen editor is displayed.
`nil`	The Modgen editor could not be displayed.

Example

To create a new Modgen constraint with the currently selected schematic instances as members:

```
mgCreateOrEdit(schemCV nil)
```

To edit an existing Modgen in the layout cellview and update the constraint and the figGroup:

```
mgCreateOrEdit(layCV modgenConstraint)
```

mgClearDummyLibCellAndParamAlias

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

Description

Clears all Modgen dummy library, cell, and parameter name aliases.

Arguments

None

Value Returned

`t`	All Modgen dummy library, cell, and parameter name aliases were deleted.
`nil`	Modgen dummy library, cell, and parameter name aliases could not be deleted.

Examples

Deletes all Modgen dummy library, cell, and parameter name aliases.

```
mgClearDummyLibCellAndParamAlias()
=> t
```

mgDeleteAllBodyContacts

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

Description

Deletes all body contacts of the Modgen when the current window is the Modgen edit window.

Arguments

None

Value Returned

`t`	Body contacts are deleted.
`nil`	Body contacts are not deleted.

Example

Deletes all body contacts in the current Modgen.

```
mgDeleteAllBodyContacts()
```

mgDeleteAllDummies

```
mgDeleteAllDummies(
     [ d_mgID ]
     )
     => t / nil
```

Description

Deletes all dummies from the specified Modgen.

Arguments

d_mgID	The figGroupId corresponding to the Modgen from which all dummies are to be deleted. If not specified, the figGroupId of the Modgen that is in the Modgen edit mode in the active window is considered.

Value Returned

t	Dummies were deleted.
nil	Dummies were not deleted.

Example

Deletes all dummies from the specified Modgen.

```
modgenid=leGetEditFigGroup()
    when(modgenid~>type=="modgen"
    mgDeleteAllDummies()
)
```

mgDeleteAllDummyRowColumnCB

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

Description

Deletes all dummy rows and columns of the Modgen when the current window is the Modgen Editing window.

Arguments

None

Value Returned

`t`	All dummy rows and columns are deleted from the Modgen.
`nil`	Dummy rows and columns could not be deleted.

Example

Deletes all dummy rows and columns in the current Modgen.

```
modgenid = leGetEditFigGroup()
        when(and(modgenid modgenid~>type=="modgen")
        mgDeleteAllDummyRowColumnCB()
            )
```

mgDeleteAllEmptyRowColumnCB

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

Description

Deletes all empty rows and columns of the Modgen when the current window is the Modgen Editing window.

Arguments

None

Value Returned

`t`	All empty rows and columns are deleted from the Modgen.
`nil`	Empty rows and columns could not be deleted.

Example

Deletes all empty rows and columns in the current Modgen.

```
modgenid = leGetEditFigGroup()
    when(and(modgenid modgenid~>type=="modgen")
    mgDeleteAllEmptyRowColumnCB()
    )
```

mgDeleteCB

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

Description

Deletes the selected dummy devices or body contacts.

Arguments

None

Value Returned

`t`	The selected body contacts or dummy devices are deleted.
`nil`	The selected body contacts or dummy devices are not deleted.

Example

Deletes selected dummies and body contacts.

```
mgDeleteCB()
```

mgDeleteEmptyRowColumnCB

```
mgDeleteEmptyRowColumnCB(
     t_type
     x_rowColNum
     )
     => nil
```

Description

Interactively deletes an empty row or column in Modgen Editor.

Arguments

t_type	Deletes an empty row or column. Valid values: `row` or `col`
x_rowColNum	Specifies the row or column number.

Examples

Deletes row `0` if it is empty.

```
mgDeleteEmptyRowColumnCB("row" 0)
```

Deletes column `1` if it is empty.

```
mgDeleteEmptyRowColumnCB("col" 1)
```

mgDestroyMatchGroupOnObject

```
mgDestroyMatchGroupOnObject(
     d_dbObjectId
     )
     => t / nil
```

Description

Destroys the matchGroups that include the specified database object.

Arguments

d_dbObjectId	The database ID of the object.

Value Returned

t	The matchGroups that included the specified database object were destroyed.
nil	Could not destroy the matchGroups that included the specified database object.

Example

Finds the trunk that is present on the second channel of a Modgen. Destroys all the matchGroups that include that trunk:

```
when(window = hiGetCurrentWindow()
    when(cellView = geGetEditCellView(window)
    when(mgId=car(exists(fg cellView~>figGroups fg~>type=="modgen"))
        when(hTrunks = mgGetChannelTrunks(mgId 2 t)
            if(mgDestroyMatchGroupOnObject(car(hTrunks)) then
                println("Matched Group that had been associated with object has
been destroyed")
            else
                println("Matched Group that is associated with object has not been
destroyed or")
                println("no matched group associated with object")
                )))))))
```

mgEditGuardRingCB

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

Description

Displays the Guard Ring Options form for creating and modifying the guard ring for the current Modgen.

Arguments

None

Value Returned

`t`	The Guard Ring Options form is displayed.
`nil`	The Guard Ring Options form is not displayed.

Example

Displays the Guard Ring Options form for the current Modgen.

```
mgEditGuardRingCB()
```

mgExecNoConObs

```
mgExecNoConObs(
     g_constraintID
     [ l_arguments ]
     )
     => t / nil
```

Description

Disables the Modgen constraint observer and evaluates the expressions in the specified sequence. Next, the function updates the constraint with any changes encountered while evaluating the expressions. After the process is complete, the Modgen constraint observer is re-enabled. This function helps change geometries or parameters of a Modgen or its instances without requiring the Modgen to enforce any consistency constraints on the resulting geometries. For example, increasing the numRows Modgen parameter using this function will cause the Modgen to have more rows, but will not rearrange the Modgen instances to fit in the new number of rows. This function is a defmacro.

Arguments

g_constraintID	List of Modgen constraint IDs that need to be updated after evaluating the specified expression.
l_arguments	List of one or more expressions that need to be evaluated in an observer-free environment.

Value Returned

t	The specified expression was evaluated and Modgen constraints were updated accordingly.

Examples

To get the Modgen constraint ID, retrieve the Modgen's figGroup ID by selecting one instance in the Modgen and executing the following in the CIW:

```
instanceID = car(geGetSelectedSet())
```

Retrieve the instances containing figGroup from the instId by executing the following:

```
figroupID = inst->figGroup
```

Retrieve the Modgen constraint ID from the figGroup ID by executing the following:

```
constraintID = mgGetConstraintFromFG(figroupID)
```

To call the following procedure in an observer-free environment (assuming that “|M1” is a member of the Modgen), run the following:

```
procedure(test(con)
ciConRemoveMembers(con list(list("|M1" 'inst))) ;
```

Run the following:

```
mgExecNoConObs(constraintID test(constraintID))
```

mgExitCB

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

Description

Closes the Modgen editing environment.

Arguments

None

Value Returned

`t`	The Modgen editing environment is closed.
`nil`	The Modgen editing environment is not closed.

Example

Closes the Modgen editing environment.

```
mgExitCB()
```

mgEvalInBackgroundModgen

```
mgEvalInBackgroundModgen(
     d_schCV
     o_schCon
     s_callback
     [ g_updateSrcConstraint ]
     )
     => l_userCallbackReturnVal / nil
```

Description

Creates a Modgen scratch cellview for the specified schematic Modgen constraint, evaluates the specified user callback in the Modgen context, and then closes and deletes the scratch cellview.

Arguments

d_schCV	The schematic cellview.
o_schCon	The schematic Modgen constraint for which a Modgen scratch cellview needs to be created.
s_callback	The user callback to be executed in a background Modgen.
g_updateSrcConstraint	Specifies whether the schematic Modgen constraint should be updated with any changes that could have taken place in the layout Modgen constraint during execution of the user callback. Default value is `t`.

Value Returned

l_userCallbackReturnVal	Return values from `callback(`*mgId layCon*`)`.
`nil`	The command was unsuccessful.

Example

Creates a Modgen scratch cellview for the specified schematic Modgen constraint, evaluates it, closes it, and deletes it.

```
procedure(getRows(mgId conId)
    caddr(car(exists(x ciConListParams(conId) car(x) == "numRows")))
)
procedure(getCols(mgId conId)
    caddr(car(exists(x ciConListParams(conId) car(x) == "numCols")))
)
procedure(setRowsGetColsUpdate(mgId conId)
    ciConUpdateParams(conId list(list("numRows" 4))); this sets "numCols" to 4
    caddr(car(exists(x ciConListParams(conId) car(x) == "numCols")))
)
procedure(setRowsGetColsNoUpdate(mgId conId)
  ; disable Modgen constraint observer while setting "numRows" to 4
  mgExecNoConObs(conId
    ciConUpdateParams(conId list(list("numRows" 4)))
    caddr(car(exists(x ciConListParams(conId) car(x) == "numCols"))) ; "numCols"
will not change because the observer was not running when "numRows" was set to 4
)
schCV = geGetEditCellView()
scache = ciCacheGet(schCV)
ciConCreate(scache 'modgen ?members list(list("PM1" 'inst) list("PM2" 'inst)))
schCon = car(scache->constraints)
mgEvalInBackgroundModgen(schCV schCon 'getRows nil)
> 1
mgEvalInBackgroundModgen(schCV schCon 'getCols nil)
> 8
mgEvalInBackgroundModgen(schCV schCon 'setRowsGetColsUpdate)
> 4
mgEvalInBackgroundModgen(schCV schCon 'setRowsGetColsNoUpdate)
> 4
```

mgFGREnterHandEdit

```
mgFGREnterHandEdit(
     d_modgenId
     )
     => t / nil
```

Description

Enters the hand edit mode for the fluid guard rings in the specified Modgen. In this mode, the fluid guard rings are not regenerated each time the Modgen is updated. Therefore, all customizations remain. The guard rings that are not edited in the hand edit mode are regenerated, and all manual edits are lost. Use `mgFGRExitHandEdit` to exit the hand edit mode.

Arguments

d_modgenId	A Modgen figGroup, storage group, or constraint ID

Value Returned

`t`	The Modgen fluid guard ring is in the hand edit mode.
`nil`	The command was unsuccessful.

Example

In the Modgen editing mode, enters the hand edit mode (if not already in it) for the Modgen-in-place.

```
mgFig = leGetEditFigGroup()
unless(mgFGRIsHandEdit(mgFig)
  mgFGREnterHandEdit(mgFig)
)
```

mgFGRExitHandEdit

```
mgFGRExitHandEdit(
     d_modgenId
     [ g_showForm ]
     )
     => t / nil
```

Description

Exits the hand edit mode for the fluid guard rings in the specified Modgen. Unlike the hand edit mode, the fluid guard rings will now be regenerated each time the Modgen is updated. To enter the hand edit mode, use `mgFGREnterHandEdit`.

Arguments

d_modgenId	A Modgen figGroup, storage group, or constraint ID
g_showForm	Controls the display of the Create Guard Ring form when you exit the hand edit mode. The default value is `t`.

Value Returned

`t`	The hand edit mode for the fluid guard ring has been exited.
`nil`	The command was unsuccessful.

Example

In the Modgen editing mode, exit the hand edit mode (if not already) for the Modgen-in-place

```
mgFig = leGetEditFigGroup()
when(mgFGRIsHandEdit(mgFig)
  mgFGRExitHandEdit(mgFig)
)
```

mgFGRIsHandEdit

```
mgFGRIsHandEdit(
     d_modgenId
     )
     => t / nil
```

Description

Checks the status of the hand edit mode of the fluid guard rings in the specified Modgen.

Arguments

d_modgenId	A Modgen figGroup, storage group, or constraint ID

Value Returned

t	The hand edit mode of fluid guides rings is turned on for the specified Modgen.
nil	The hand edit mode of fluid guides rings is turned off for the specified Modgen.

Example

In the Modgen editing mode, checks the status of hand edit mode for the Modgen in place.

```
mgFig = leGetEditFigGroup()
when(mgFGRIsHandEdit(mgFig)
  mgFGRExitHandEdit(mgFig)
)
```

mgFlattenModgens

```
mgFlattenModgens(
     d_cellViewID
     ld_modgenID
     )
     => t / nil
```

Description

Flattens the specified Modgens in the specified cellview. When a Modgen is flattened, its constraint is removed, it is ungrouped, and all geometry created by it remains ungrouped in the layout cellview.

When the Modgen constraint is removed, the Modgen grouping is no longer recognized by Virtuoso, which means that it can be modified interactively and by the automatic tools. Do not use this function if you want to maintain the grouping. Disable the constraint instead.

Arguments

d_cellViewID	Layout cellview in which modgens are to be flattened.
ld_modgenID	List of Modgen IDs to be flattened. Default value: `nil` If you do not specify one or more Modgen IDs, all the modgens in the specified cellview are flattened.

Value Returned

`t`	The specified Modgens were flattened.
`nil`	The specified Modgens were not flattened.

Example

Flattens Modgens in the current cellview.

```
mgFlattenModgens(geGetEditCellview())
```

mgFlipHorizontalCB

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

Description

Horizontally flips the selected instance. To flip an instance, at least one instance should be selected. After you flip the instance, the orientation of the selected instance changes to reflect the flip.

Arguments

None

Value Returned

t	Flips the instance horizontally.
nil	The current window cellview ID is invalid.

Example

Flips selected instance horizontally.

```
mgFlipHorizontalCB()
```

mgFlipVerticalCB

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

Description

Vertically flips the selected instance. To flip an instance, at least one instance should be selected. After you flip the instance, the orientation of the selected instance changes to reflect the flip.

Arguments

None

Value Returned

`t`	Flips the instance vertically.
`nil`	The current window cellview ID is invalid.

Example

Flips the selected instance vertically.

```
mgFlipVerticalCB()
```

mgGetBoxLPPArea

```
mgGetBoxLPPArea(
     d_cellViewID
     l_bBox
     tx_layerName
     t_purposeList
     )
     => x_TotalModgenArea / nil
```

Description

Calculates the total area on the specified layer within the specified bounding box. If you specify a Modgen ID as the *l_bBox* parameter, then the total Modgen area is calculated and displayed.

Arguments

d_cellViewID	The database ID of the cellview.
l_bBox	The bounding box for which area needs to be calculated. Here, you can specify the Modgen ID to calculate the total Modgen area.
tx_layerName	Layer on which the total area for bBox needs to be calculated.
t_purposeList	Purposes for which total area for bBox needs to be calculated.

Value Returned

x_TotalbBoxArea	Total area on the specified layer within the specified bounding box.
`nil`	The operation failed. The function will return `nil` if the total bBox area could not be calculated.

Examples

The following example calculates the "`Metal1`" layer density inside the editing Modgen:

```
modgenId = leGetEditFigGroup()
```

The following example calculates the total `metal1` area inside the Modgen bounding box:

```
metal1Area = mgGetBoxLPPArea(geGetEditCellview() modgenId~>bBox "Metal1"
list("drawing" "pin"))
```

The following example calculates total Modgen area:

```
modgenArea = adleBBoxArea(modgenId~>bBox)
```

The following example divides `metal1Area` and `modgenArea` to get the `metal1Density` inside Modgen:

```
metal1Density = metal1Area/modgenArea
```

mgGetChannelTrunks

```
mgGetChannelTrunks(
     d_mgCon
     n_channel
     g_isHoriz
     )
     => l_topologyTrunks / nil
```

Description

Returns a list of topology trunks that belong to the specified channel.

Arguments

d_mgCon	Modgen constraint ID.
n_channel	Channel number.
g_isHoriz	A boolean value that specifies whether the trunk is horizontal.

Value Returned

l_topologyTrunks	List of topology trunks that belong to the specified channel.
nil	The list of topology trunks could not be retrieved.

Example

Returns a list of topology trunks that belong to the given channel.

```
mgId = car(geGetEditCellView()~>figGroups)
mgCon = mgGetConstraintFromFG(mgId)
printf("Number of trunks in horizontal channel 1 = %d\n"
length(mgGetChannelTrunks(mgId 1 t)))
printf("Number of trunks in vertical channel 3 = %d\n"
length(mgGetChannelTrunks(mgId 3 nil)))
```

mgGetColumnRoutingChannelWidth

```
mgGetColumnRoutingChannelWidth(
     d_constraintId
     x_channelIndex
     )
     => f_channelWidth / nil
```

Description

Returns the routing channel width of the specified Modgen column.

Arguments

d_constraintId	Modgen constraint ID.
x_channelIndex	Channel index, which indicates the Modgen column for which the channel width needs to be displayed.

Value Returned

f_channelWidth	The channel width of the specified Modgen column.
nil	The channel width could not be retrieved.

Example

The following example returns the routing channel width for the second routing channel column:

```
c = car(ciCacheGet(geGetEditCellView())->constraints)
when(c
    mgSetColumnRoutingChannelWidth(c 1 0.5)
    mgSetColumnRoutingChannelWidth(c 2 0.6)
    channelWidth = mgGetColumnRoutingChannelWidth(c 2)
    )
```

mgGetConstraintFromFG

```
mgGetConstraintFromFG(
     d_modgenID
     )
     => d_constraint_ID / nil
```

Description

Returns the database ID of the Modgen constraint for the specified Modgen figGroup ID.

Arguments

d_modgenID	List of Modgen figGroup IDs.

Value Returned

d_constraint_ID	The Modgen constraint ID of associated Modgen figGroup.
`nil`	The operation failed. The function will return `nil` if you provide an instance ID, or if you provide a figGroup ID that is not Modgen related.

Example

Returns the database ID of the Modgen constraint.

```
mgGetConstraintFromFG(modgenid)
```

mgGetDummyLibCellAndParamAlias

```
mgGetDummyLibCellAndParamAlias(
     )
     => l_ActiveLibCells_and_params l_DummyLibCells_and_params / nil
```

Description

Returns all currently registered Modgen dummy library and cell mapping and parameter name mapping lists.

Arguments

None

Value Returned

l_ActiveLibCells_and_params
l_DummyLibCells_and_params
	All currently registered Modgen Dummy library, cell mapping and parameter name mapping in the following format:
	`list(`
	`list(`
	`list(list(libNameA cellNameA) list(paramA1 paramA2 .... paramAN))`
	`list(list(libNameB cellNameB) list(paramB1 paramB2 .... paramBN))`
	`)`
	`list(`
	`list(list(libNameC cellNameC) list(paramC1 paramC2 .... paramCN))`
	`list(list(libNameD cellNameD) list(paramD1 paramD2 .... paramDN))`
	`)`
	`...`
	`list(`
	`list(list(libNameY cellNameY) list(paramY1 paramY2 .... paramYN))`
	`list(list(libNameZ cellNameZ) list(paramZ1 paramZ2 .... paramZN))`
	`)`
	`)`

`nil`	There are no registered Modgen dummy library and cell mapping and parameter name mappings.

Example

Retrieves a list of all currently registered Modgen dummy library and cell mapping and parameter name mapping lists.

```
mgSetDummyLibCellAndParamAlias(list(list("libNameA" "cellNameA")
list("paramNameA1" "paramNameA2")) list(list("libNameB" "cellNameB")
list("paramNameB1" "paramNameB2")))
=> t
> mgGetDummyLibCellAndParamAlias()
=> ((("libNameA" "cellNameA")
        ("paramNameA1" "paramNameA2")
    )
    (("libNameB" "cellNameB")
        ("paramNameB1" "paramNameB2")
    )
    )
)

mgClearDummyLibCellAndParamAlias()
=> t

mgGetDummyLibCellAndParamAlias()
=> nil
```

mgGetDummyRefDirectionParams

```
mgGetDummyRefDirectionParams(
     [ t_libName ]
     [ t_cellName ]
     [ t_paramName ]
     )
     => l_registeredDummyDirectionalParams / nil
```

Description

Returns the first set of registered Modgen dummy directional parameters that match the specified library, cell, and parameter names.

Arguments

t_libName	Name or the library for which registered Modgen dummy directional parameters are to be retrieved. If not specified, all registered directional parameters are listed.
t_cellName	Name of the cell for which registered Modgen dummy directional parameters are to be retrieved. If not specified, all cell directional parameters that are registered for the given library are listed.
t_paramName	Name of the directional parameter for which the direction should be retrieved. The function returns the direction. If the parameter name is not specified, the function returns a list of parameters registered to the first match of the library and cell name.

Value Returned

l_registeredDummyDirectionalParams	Depending on the specified arguments, the library, cell, and parameters of the first registered match are returned. For example, if the library, cell, and parameter names are specified, only the direction of the parameter in the first registered match of the cell is returned.
`nil`	There are currently no registered directional parameters.

Example

Registers Modgen dummy directional parameters for different libraries and directions and then shows various ways to retrieve the parameters.

```
mgRegisterDummyParamsRefDirection("libName1" "NcellName" "Horizontal"
list("width" "connectGates"))
mgRegisterDummyParamsRefDirection("libName1" "NcellName" "Vertical"
list("fingers" "length"))


mgRegisterDummyParamsRefDirection("libName1" "P*" "Horizontal" list("width"
"connectSD"))
mgRegisterDummyParamsRefDirection("libName1" "P*" "Vertical" list("fingers"
"length"))


mgRegisterDummyParamsRefDirection("libName1" "N*" "Horizontal" list("width"
"connectSD"))
mgRegisterDummyParamsRefDirection("libName1" "N*" "Vertical" list("fingers"
"length"))


mgRegisterDummyParamsRefDirection("libName2" "*cellName" "Horizontal"
list("width" "connectSD"))
mgRegisterDummyParamsRefDirection("libName2" "*cellName" "Vertical"
list("fingers" "length"))


;; Retrieves the direction for this parameters
mgGetDummyRefDirectionParams("libName1" "PcellName" "width")
"Horiztonal"


;; Retrieves a DPL with list of all parameters for the given library and cell name
mgGetDummyRefDirectionParams("libName1" "NcellName2")
(nil Vertical ("fingers" "length") Horizontal ("connectGates" "width"))


;; Retrieves a list of DPLs for the given library name
```

```
mgGetDummyRefDirectionParams("libName1")
(("NcellName"
    (nil Vertical
    ("fingers" "length") Horizontal
    ("connectGates" "width")
    )
)
("P*"
    (nil Vertical
    ("fingers" "length") Horizontal
    ("connectSD" "width")
    )
)
("N*"
    (nil Vertical
    ("fingers" "length") Horizontal
    ("connectSD" "width")
    )
)
)

;; Retrieves a list of lists of DPLs for all registered library names
mgGetDummyRefDirectionParams()
(("libName2"
    (("*cellName"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
    )
)
("libName1"
    (("NcellName"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectGates" "width")
        )
        )
        ("P*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
        ("N*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
    )
)
)
("libName1"
    (("NcellName"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectGates" "width")
        )
```

```
        )
        ("P*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
        ("N*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
    )
)
)
```

Related Topics

mgRegisterDummyParamsRefDirection

mgUnregisterDummyParamRefDirection

mgGetGuardRingConnectObject

```
mgGetGuardRingConnectObject(
     d_constraintId
     d_modgenId
     )
     => d_grObjectId / nil
```

Description

Returns the guard ring object for the topology connection for the specified Modgen.

Arguments

d_constraintId	Constraint ID.
d_modgenId	Modgen ID for which the guard ring object needs to be retrieved.

Value Returned

d_grObjectId	The name of the guard ring object.
nil	The guard ring object could not be retrieved.

Example

Here, `mgId` holds the Modgen figGroup ID and `cid` holds the Modgen constraint ID. If the Modgen guard ring is an MPP, then the object holds the shape ID that is used to connect the guard ring. If it is a fluid guard ring, then the object holds the instTerm that is used to connect the guard ring.

```
object = mgGetGuardRingConnectObject(cid mgId)
```

mgGetIsLocalTrunk

```
mgGetIsLocalTrunk(
     d_trunkId
     )
     => t / nil
```

Description

Checks whether the specified trunk is local. Trunks inside a Modgen are called local trunks. These trunks can be connected only within the same Modgen, and not from outside the Modgen.

Arguments

d_trunkId	Database ID of the topology trunk.

Value Returned

`t`	The specified topology trunk is local.
`nil`	The specified topology trunk is not local.

Example

Creates a topology trunk and checks whether the trunk is local:

```
cv = geGetEditCellView()
net = dbFindNetByName(cv, "3")
myTopo = dbCreateTopology("myTopo" net)
myTrunk = dbCreateTrunk("myTrunk" myTopo "horizontal")
mgSetIsLocalTrunk(myTrunk t)
when(mgGetIsLocalTrunk(myTrunk)
    printf("Topology trunk is local.\n")
)
```

mgGetLayerSpacing

```
mgGetLayerSpacing(
     d_instance1
     d_instance2
     t_referenceLayer
     t_direction
     x_spacing
     )
     => x_effectiveSpacing / nil
```

Description

Returns the effective spacing between two instances, given the spacing between a reference layer inside the instances. For example, if the reference layer is `Oxide_thk drawing`, then `x_spacing` would indicate the spacing between the bounding box of the `Oxide_thk` layer inside the instances. The `t_direction` argument specifies the position of the instances relative to each other, either `horizontal` (next to each other) or `vertical` (above and below each other).

Arguments

d_instance1	The database ID of the first instance.
d_instance2	The database ID of the second instance.
t_referenceLayer	The reference layer inside the instances. The reference layer is a string in the form <layer> <purpose>.
t_direction	The direction in which spacing is measured. The direction can be horizontal or vertical.
x_spacing	The spacing between a reference layer inside the instances.

Value Returned

x_effectiveSpacing	The effective spacing between two instances specified in the function.
`nil`	The operation failed.

Example

Returns the effective spacing between the two instances.

```
mgGetLayerSpacing(
                inst1
                inst2
                "Oxide_thk drawing"
                "horizontal"
                0.5
                )
            => -0.3
```

mgGetMatchGroupMembers

```
mgGetMatchGroupMembers(
     d_dbObjectId
     )
     => t / nil
```

Description

Displays a list of matchGroup members that belong to the same matchGroup as the specified database object *d_dbObjectId*.

Arguments

d_dbObjectId	The database ID of the object for which the associated matchGroup members need to be retrieved.

Value Returned

t	The matchGroups members were listed.
nil	The command was unsuccessful.

Example

Prints the number of members that are in the matchGroup associated with the first horizontal trunk in channel 2 of the Modgen:

```
when(window = hiGetCurrentWindow()
    when(cellView = geGetEditCellView(window)
        when(mgId = car(exists(fg cellView~>figGroups fg~>type=="modgen"))
            when(hTrunks = mgGetChannelTrunks(mgId 2 t)
            mems = mgGetMatchGroupMembers(car(hTrunks)
            if(mems then
                printf("Object is part of a Matched Group that has %d members"
length(mems))
            else
                println("Object is not part of a Matched Group")
                )
            )
        )
    )
)
)
```

mgGetModgenConstraintFromTopology

```
mgGetModgenConstraintFromTopology(
     d_tpObjectId
     )
     => d_modgenId / nil
```

Description

Returns the Modgen constraint ID that is associated with the specified topology object.

Arguments

d_tpObjectId	A topology object ID.

Value Returned

d_modgenId	The database ID of the Modgen constraint object.
nil	The command was unsuccessful.

Example

In this example, `topo` is a topology pattern ID that has been added to a Modgen using `mgAddTopologyToModgen`, and `cid` is the Modgen constraint ID of the Modgen that contains that topology pattern.

```
cid = mgGetModgenConstraintFromTopology(topo)
```

mgGetModgenFigGroupFromTopology

```
mgGetModgenFigGroupFromTopology(
     d_tpObjectId
     )
     => d_figGroupId / nil
```

Description

Retrieves the Modgen figGroup object that is associated with the specified topology object.

Arguments

d_tpObjectId	A topology object ID.

Value Returned

d_figGroupId	The database ID of the Modgen figGroup.
nil	The command was unsuccessful.

Example

In this example, `topo` is a topology pattern ID that has been added to a Modgen using `mgAddTopologyToModgen`. `mgId` holds the Modgen figGroup ID of the Modgen that contains that topology pattern.

```
mgId = mgGetModgenFigGroupFromTopology(topo)
```

mgGetModgenFGFromConstraint

```
mgGetModgenFGFromConstraint(
     d_cellViewID
     d_constraintName
     )
     => d_figGroupID / nil
```

Description

Returns the database ID of the Modgen figGroup for a specified Modgen constraint ID in a specified cellview.

Arguments

d_cellViewID	The cellview containing the Modgen constraint for which you want to retrieve the figGroup ID.
d_constraintName	Modgen constraint name.

Value Returned

d_figGroupID	The database ID of the figGroup associated with the Modgen constraint.
`nil`	The operation failed. The function will return `nil` if you provide an alignment constraint name or any random string.

Example

Returns the database ID of the Modgen figGroup for a specified Modgen constraint ID.

```
mgGetModgenFGFromConstraint(lcv ciConGetName(cid))
```

mgGetRegisteredDummyNetProc

```
mgGetRegisteredDummyNetProc(
     )
     => l_dummyNetProc / nil
```

Description

Returns a list of `dummyNet` procedures that are currently registered.

Arguments

None

Value Returned

`l_dummyNetProc`	A list of registered `dummyNet` procedures. Example: `list("'myFunc1" "'myFunc2" ...)`
`nil`	There are currently no registered `dummyNet` procedures.

Example

In the following example, a `dummyNet` procedure `'GateChangesForDeviceType` is defined and then registered. Running the `mgGetRegisteredDummyNetProc` function returns `'GateChangesForDeviceType`. This `dummyNet` procedure is then unregistered. If you now run the `mgGetRegisteredDummyNetProc` function, it returns `nil`:

```
procedure( GateChangesForDeviceType(instId termId)
  let((net)
    when(exists(name ciGetTermNames("gate") name==termId~>name)
      when(ciIsDevice(instId "nmos")
        net = dbMakeNet(instId~>cellView "vdda_nmos")
      )
      when(ciIsDevice(instId "pmos")
        net = "vdda_pmos"
      )
    )
    net
  )
```

```
)

mgRegisterDummyNetProc('GateChangesForDeviceType)
mgGetRegisteredDummyNetProc()
=> list("'GateChangesForDeviceType")
mgUnRegisterDummyNetProc('GateChangesForDeviceType)
mgGetRegisteredDummyNetProc()
=> nil
```

mgGetRegisteredDummyOverrideParameters

```
mgGetRegisteredDummyOverrideParameters(
     [ t_libName ]
     [ t_cellName ]
     [ t_paramType ]
     )
     => l_registeredDummyOverrides / nil
```

Description

Returns the first set of registered Modgen dummy override parameters that match the specified library and cell names. If the library name is not specified, the function returns all registered overrides. If the cell name is not specified, the function returns all cell overrides that are registered for the given library.

Arguments

t_libName	Name or the library for which registered Modgen dummy override parameters are to be retrieved. If not specified, all registered overrides are listed.
t_cellName	Name or the cell for which registered Modgen dummy override parameters are to be retrieved. If not specified, all cell overrides that are registered for the given library are listed.
t_paramType	Sets the parameter type to one of the following: `supply`, `signal`, or `all`. The default value is `supply`.

Value Returned

l_registeredDummyOverrides

If library and cell names are specified, the parameters of the first registered match is returned.

If cell name is not specified, all cell overrides that are registered for the given library are returned.

If neither library or cell names are specified, all registered Modgen dummy override parameters are returned.

`nil` There are currently no registered `dummyNet` procedures.

Examples

Registers overrides for three different libraries and retrieves the overrides by specifying only the library name, both library and cell names, and neither library or cell names.

```
mgRegisterDummyOverrideParameters("libName1" "cellname1" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE")))
mgRegisterDummyOverrideParameters("libName1" "cellName2" list(list(
"boolStrParam1" "FALSE") list("boolStrParam2" "FALSE")))
mgRegisterDummyOverrideParameters("libName2" "cell*2" list(list( "boolStrParam3"
"FALSE") list("boolStrParam4" "FALSE")))
mgRegisterDummyOverrideParameters("libName3" "cellName3" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
```

Returns all registered Modgen dummy override parameters in the specified library.

```
mgGetRegisteredDummyOverrideParameters("libName1" 'all)
=> (("cellname1"
    (("boolStrParam2" "TRUE")
        ("boolStrParam1" "TRUE")
    )
    )
    ("cellName2"
    (("boolStrParam2" "FALSE")
        ("boolStrParam1" "FALSE")
    )
    )
    )
```

Returns all registered Modgen dummy override parameters in the specified library and cell.

```
mgGetRegisteredDummyOverrideParameters("libName2" "cellAnyStringHere2" 'signal)
=> (("boolStrParam2" "FALSE")
    ("boolStrParam1" "FALSE")
    )
```

Returns all registered Modgen dummy override parameters.

```
mgGetRegisteredDummyOverrideParameters()
=> (("libName3"
    (("cell[nN]ame3"
        (("boolStrParam4" "FALSE")
            ("boolStrParam3" "FALSE")
        )
        )
    )
    )
```

```
    ("tsmcN5"
    (("analog_cell"
        (("cellType" "Dummy_cell"))
        )
        ("hp_analog_cell"
        (("cellType" "Dummy_cell"))
        )
    )
    )
    ("libName1"
    (("cellname1"
        (("boolStrParam2" "TRUE")
            ("boolStrParam1" "TRUE")
        )
        )
        ("cellName2"
        (("boolStrParam2" "FALSE")
            ("boolStrParam1" "FALSE")
        )
        )
    )
    )
    ("tsmcN3"
    (("analog_cell"
        (("pdk_dmy_switch" "ON"))
        )
    )
    )
    ("libName2"
    (("cell*2"
        (("boolStrParam4" "FALSE")
            ("boolStrParam3" "FALSE")
        )
        )
    )
    )
)
```

mgGetRegUserProc

```
mgGetRegUserProc(
     s_userFunction
     )
     => user_functions / nil
```

Description

Accepts a user function symbol and returns the names of the user functions that are registered corresponding to the specified user function symbol.

Arguments

s_userFunction	Represents a user function symbol. Valid user functions are: ■ `'mgAbutGetEnvProc`: Returns the current state of user-defined abutment types. ■ `'mgAbutSetEnvProc`: Sets the current state of user-defined abutment types. ■ `'mgUserShapeProc`: Registers as user shape callbacks. ■ `'mgPrePlacementProc`: Is called before placement in the Modgen. ■ `'mgPlacementCheckProc`: Is called after placement to either modify the constraint and rerun placement or accept the placement and proceed with routing. ■ `'mgGetIdenticalGRDefsProc`: Returns the guard ring definition, unit cell parameters, and all associated callbacks. ■ `'mgRouterUserShapeProc`: Creates user-defined shapes inside the Modgen after routing. ■ `'mgSurroundDefaultsProc`: Sets the default size of devices that surround adjoining Modgen instances when creating identical guard rings.

Value Returned

user_functions	Names of the user functions that are registered corresponding to the specified user function symbol.
`nil`	No user functions were registered.

Examples

Returns the names of the user functions that are registered for to the specified user function symbols.

```
mgGetRegUserProc('mgAbutGetEnvProc)
mgGetRegUserProc('mgAbutSetEnvProc)
mgGetRegUserProc('mgUserShapeProc)
```

```
mgGetRegUserProc('mgPrePlacementProc)
```

mgGetRowRoutingChannelWidth

```
mgGetRowRoutingChannelWidth(
     d_constraintId
     x_channelIndex
     )
     => f_channelWidth / nil
```

Description

Returns the routing channel width of the specified Modgen row.

Arguments

d_constraintId	Modgen constraint ID.
x_channelIndex	Channel index, which indicates the Modgen row for which the channel width needs to be displayed.

Value Returned

f_channelWidth	The channel width of the specified Modgen row.
nil	The channel width could not be retrieved.

Example

The following example returns the routing channel width of the second channel row:

```
c = car(ciCacheGet(geGetEditCellView())->constraints)
when(c
    mgSetRowRoutingChannelWidth(c 1 0.5)
    mgSetRowRoutingChannelWidth(c 2 0.6)
    channelWidth = mgGetRowRoutingChannelWidth(c 2)
    )
```

mgGetStrapDirection

```
mgGetStrapDirection(
     d_strapId
     )=> d_dir / nil
```

Description

Returns the direction of the specified strap. This function is valid if the strap has a single strap object. Otherwise, the router determines the strap direction.

Arguments

d_strapId	Database ID of the strap topology for which the direction needs to be retrieved.

Value Returned

d_dir	Direction of the strap with the specified ID.
nil	The command was unsuccessful.

Example

Returns the direction of the specified strap.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
println(mgGetStrapDirection(strap))
```

mgGetStrapHasDirection

```
mgGetStrapHasDirection(
     d_strapId
     ) => t / nil
```

Description

Determines whether the strap direction is set for the specified strap.

Arguments

d_strapId	Database ID of the strap topology.

Value Returned

t	The strap direction is set.
nil	The strap direction is not set.

Example

Determines whether the strap direction is set for the specified strap.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
when(mgGetStrapHasDirection(strap)
    println(mgGetStrapDirection(strap))
)
```

mgGetStrapHasOffset

```
mgGetStrapHasOffset(
     d_strapId
     )
     => t / nil
```

Description

Determines whether the specified strap has an associated offset value.

Arguments

d_strapId	Database ID of the strap topology.

Value Returned

t	The strap has an offset value specified.
nil	The strap does not have an offset value.

Example

Determines whether the specified strap has an associated offset value.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
when(mgGetStrapHasOffset(strap)
    println(mgGetStrapHasOffset(strap))
    )
```

mgGetStrapOffset

```
mgGetStrapOffset(
     d_strapId
     )
     => n_offset / nil
```

Description

Returns the strap offset attribute, which represents the distance from the bottom or left of the bBox of the `strapObject`.

Arguments

d_strapId	Database ID of the strap topology.

Value Returned

n_offset	The offset value of the specified strap.
`nil`	The strap does not have an offset value.

Example

Returns the strap offset attribute of the specified strap topology.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
println(mgGetStrapOffset(strap))
```

mgGetStrapLongOffset1

```
mgGetStrapLongOffset1(
     d_strapId
     )
     => n_longOffset1 / nil
```

Description

Returns the long offset attribute of the specified strap for the bottom or left edge. This value determines how far beyond the strap object that the strap can extend.

Arguments

d_strapId	Database ID of the strap topology.

Value Returned

n_longOffset1	The long offset attribute of the specified strap.
nil	The strap does not have an offset value.

Example

Returns the long offset attribute of the bottom or left edge.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
println(mgGetStraLongpOffset1(strap))
```

mgGetStrapLongOffset2

```
mgGetStrapLongOffset2(
     d_strapId
     )
     => n_longOffset2 / nil
```

Description

Returns the long offset attribute of the specified strap for the top or right edge. This value determines how far beyond the strap object that the strap can extend.

Arguments

d_strapId	Database ID of the strap topology.

Value Returned

n_longOffset2	The long offset attribute of the specified strap.
nil	The strap does not have an offset value.

Example

Returns the long offset attribute of the top or right edge.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
println(mgGetStrapLongOffset2(strap))
```

mgGetTopologyFromModgen

```
mgGetTopologyFromModgen(
     d_modgenId
     )
     => l_tpObjects / nil
```

Description

Returns a list of topology objects that are available in the specified Modgen.

Arguments

d_modgenId	A Modgen ID

Value Returned

l_tpObjects	A list of topology objects found in the specified Modgen.
nil	The command was unsuccessful.

Example

Returns a list of topology objects that are available in the Modgen with constraint ID `cid`.

```
mgTopo = mgGetTopologyFromModgen(cid)
```

mgGetTopoTrunkShapes

```
mgGetTopologyFromModgen(
     d_topoTrunkId
     )
     => l_shapeIds / nil
```

Description

Retrieves the shapes associated with a topological trunk.

Arguments

d_topoTrunkId	Database ID of the topological trunk object.

Value Returned

l_shapeIds	List of shapes associated with the specified topological trunk.
nil	The command was unsuccessful.

Example

Returns a list of shapes for the trunk pathSeg.

```
trunk = mgGetTrunkTopo(css())
mgGetTopoTrunkShapes(trunk)
```

mgGetTopoShapes

```
mgGetTopoShapes(
     d_topoId
     )
     => l_shapeIds / nil
```

Description

Retrieves all user-generated shapes from the specified topology group object, usually a topological strap.

Arguments

d_topoId	Database ID of the topological group or strap.

Value Returned

l_shapeIds	List of user-generated shapes present in the specified topology group object.
nil	The command was unsuccessful.

Example

Returns all user-generated shapes from the specified topology group object.

```
cv = geGetEditRep()
rect = dbCreateRect(cv "Metal2" list(10:10 12:10.2))
topos = dbGetCellViewTopologies(cv)
inpTopos = car(exists(topo topos dbGetTopologyNet(topo)~>name == "INP"))
strap = car(dbGetTopologyObjects(inpTopos))
mgAddShapeToTopo(strap, rect)
mgGetTopoShapes(strap)
```

mgGetTrunkChannel

```
mgGetTrunkChannel(
     d_mgConId
     d_trunkId
     )
     => x_channel / nil
```

Description

Returns the channel number in which the specified topological trunk pattern is located.

Arguments

d_mgConId	Database ID of the Modgen constraint in which the specified topological trunk pattern is located.
d_trunkId	Database ID of the topological trunk pattern.

Value Returned

x_channel	Integer indicating the number of the row or column in which the topological trunk pattern is located in the Modgen.
nil	The topological trunk object was not found.

Example

```
cv = geGetEditRep()
mgConId = car(ciCacheListCon(ciCacheGet(cv)))

; Assume the user has selected a physical trunk (pathSeg) in the Modgen
topoTrunk = mgGetTrunkTopo(css())
channel = mgGetTrunkChannel(mgConId topoTrunk)
```

mgGetTrunkRefLayerPurpose

```
mgGetTrunkRefLayerPurpose(
     d_trunkId
     )
     => l_layerPurposePair / nil
```

Description

Returns the reference layer and purpose of a topological trunk. This function returns a list containing the layer and purpose names.

Arguments

d_trunkId	Database ID of the topological trunk for which the reference layer and purpose need to be retrieved.

Value Returned

l_layerPurposePair	The reference layer and purpose of the specified topological trunk.
nil	The layer and purpose names could not be obtained.

Example

Returns the reference layer and purpose of the given topological trunk.

```
cv = geGetEditRep()
        n = dbFindNetByName(cv "net7")
        topo = car(setof(x dbGetCellViewTopologies(cv) equal(dbGetTopologyNet(x)
n)))
        trunk = car(dbGetTopologyObjects(topo))
        mgGetTrunkRefLayerPurpose(trunk)
        ===>("Oxide_thk" "drawing")
```

mgGetTrunkRefLPPEnclosure

```
mgGetTrunkRefLPPEnclosure(
     d_instID
     t_side
     t_layerName
     t_purposeName
     )
     => x_enclosure / nil
```

Description

Returns the enclosure of a topological trunk reference layer and the purpose within an instance. For example, if the LPP of a trunk reference is `Oxide drawing`, then the function returns the distance from one side of the `Oxide` bounding box to the instance bounding box. This value is subtracted from the channel width of the topological trunk to derive the trunk-to-device spacing to the reference layer in the instances above (or to the right of) a trunk.

Arguments

d_instID	The database identifier of the instance to be used to compute the reference layer and purpose enclosure.
t_side	A string to indicate the side of the instance bounding box to use in the reference layer and purpose enclosure calculation. Valid values are: `top`, `bottom`, `left`, and `right`.
t_layerName	A string indicating the reference layer name.
t_purposeName	A string indicating the reference purpose name.

Value Returned

x_enclosure	The reference layer and purpose enclosure.
`nil`	Command did not run because of errors in the arguments.

Example

Returns the enclosure of a topological trunk reference layer and the purpose within an instance.

```
channelWidth = (numTrunks * trunkWidth + (numTrunks -1) * trunkSpacing
               + bottomTrunkToDeviceSpacing + topTrunkToDeviceSpacing
cellView = geGetEditCellView()
```

```
instance = dbFindAnyInstByName(cellView "M0")
enclosure = mgGetTrunkRefLPPEnclosure(instance "bottom" "Oxide_thk" "drawing")
channelWidth = channelWidth - enclosure
```

mgGetTrunkTopo

```
mgGetTrunkTopo(
     d_pathSeg
     )
     => l_trunkIds / nil
```

Description

Returns a list of topology trunks belonging to the specified pathSeg.

Arguments

d_pathSeg	Database ID of the pathSeg for which associated topological trunks need to be retrieved.

Value Returned

l_trunkIds	List of topology trunks belonging to the specified pathSeg.
nil	The command could not be executed.

Example

Returns a list of topology trunks.

```
mgId = car(geGetEditCellView()~>figGroups)
geomTrunks = setof(fig mgId~>figs mgGetTrunkTopo(fig)!=nil)
print("There are ")
print(length(geomTrunks))
println("pathsegs associated with topological trunks in this Modgen")
```

mgHilightEmptyRowColumnCB

```
mgHilightEmptyRowColumnCB(
     t_disp
     )
     => t / nil
```

Description

Turns on or off highlighting of empty rows and columns.

Argument

t_disp	Turns on or off highlighting of all empty rows and columns. Valid values: `show` or `hide`

Value Returned

`t`	Highlighting for empty rows was turned on/off.
`nil`	The command could not be executed.

Examples

Turns on highlighting of all empty rows and columns.

```
mgHilightEmptyRowColumnCB("show")
```

Turns off highlighting of all empty rows and columns.

```
mgHilightEmptyRowColumnCB("hide")
```

mgIsInBackAnnotation

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

Description

Checks if there are any Modgens currently running dummy backannotation. Use this function for user callbacks to allow changes to the Modgen constraint caused by backannotation, while preventing other changes to the Modgen constraint.

Argument

None

Value Returned

`t`	A Modgen is currently running dummy backannotation.
`nil`	No Modgen detected in backannotation.

Example

In the following procedure, the mgIsInBackAnnotation function is used in a template callback. If the backannotation process is running, then the dpCheck function returns true and the template constraints are permitted to be modified. Else, the function returns nil and prohibits the constraints from changing.

```
procedure(dpCheckCB(template
         @key (params nil)
         (userParams nil) )
         mgIsInBackAnnotation()
         ); procedure
```

mgIsTopologyInsideModgen

```
mgIsTopologyInsideModgen(
     d_tpObjectId
     [ d_figGroupId ]
     )
     => t / nil
```

Description

Checks whether the specified topology object belongs to a Modgen. For example, this function can be used to identify Modgen topologies or to find all topologies that are available in the Modgen specified by the optional *d_figGroupId* argument.

When the figGroup ID (optional argument) is specified, then the function checks whether the specified topology object belongs to the specified Modgen (*d_figGroupId*). When not specified, then the function checks whether the specified topology object belongs to any Modgen.

Arguments

d_tpObjectId	A topology object ID.
d_figGroupId	A Modgen figGroup ID.

Value Returned

t	The topology object belongs to a Modgen. If d_figGroupId was specified, the topology object belongs to the specified Modgen.
nil	The topology object does not belong to a Modgen.

Example

The following example returns `t` if the topology is contained in the Modgen with topology pattern ID set to `topo`:

```
mgIsTopologyInsideModgen(topo)
```

Here, the topology pattern ID topo has been added to a Modgen and the variable mgId holds the Modgen figGroup ID. The following example will return t if the topology is contained in the specific Modgen that is represented by mgId:

```
mgIsTopologyInsideModgen(topo mgId)
```

mgModgenHasTopology

```
mgModgenHasTopology(
     d_modgenId
     )
     => t / nil
```

Description

Checks whether the specified Modgen contains a topology.

Arguments

d_modgenId	A valid Modgen figGroup, storage group, or constraint ID.

Value Returned

`t`	Topology was found in the specified Modgen.
`nil`	Topology was not found in the specified Modgen.

Example

Returns `t` if the specified Modgen constraint ID (`cid`) has a topology pattern:

```
mgModgenHasTopology(cid)
```

mgObjectHasMatchGroup

```
mgObjectHasMatchGroup(
     d_dbObjectId
     )
     => t / nil
```

Description

Checks whether the specified database object is part of a matchGroup.

Arguments

d_dbObjectId	The database ID of the object to be checked.

Value Returned

t	The database object is part of one or more matchGroups.
nil	The database object is not part of any matchGroup.

Example

Checks if a the specified trunk is part of any matchGroup:

```
when(window = hiGetCurrentWindow()
    when(cellView = geGetEditCellView(window)
        when(mgId=car(exists(fg cellView~>figGroups fg~>type=="modgen"))
            when(hTrunks = mgGetChannelTrunks(mgId 2 t)
                if(mgObjectHasMatchGroup(car(hTrunks)) then
                    println("Object is part of a Matched Group")
                else
                    println("Object is not part of a Matched Group")
            )))))
```

mgRegenerateModgen

```
mgRegenerateModgen(
     d_modgenId
     )
     => t / nil
```

Description

Forces a Modgen-type group to rebuild its geometry.

Arguments

d_modgenId	A valid Modgen figGroup, storage group, or constraint ID.

Value Returned

t	The Modgen was regenerated.
nil	Regeneration of the Modgen failed.

Example

After a PDK change that changed the parameters of objects in a Modgen, the following code fragment regenerates the Modgens in a layout view to display the updated geometry:

```
when(window = hiGetCurrentWindow()
    when(cellView = geGetEditCellView(window)
        foreach(fg cellView~>figGroups
    when(fg~>type == "modgen"
        mgRegenerateModgen(fg)
        ))))
```

mgRegisterDummyNetProc

```
mgRegisterDummyNetProc(
     s_userProc
     )
     => t / nil
```

Description

Registers a function that specifies a net for a specified dummy instance ID and terminal ID.

Arguments

s_userProc	Specifies the name of a function that includes two parameters—instance ID and terminal ID—and returns a `netName` or `netId`.

Value Returned

`t`	The function was registered.
`nil`	The command was unsuccessful.

Example

The following example first defines a procedure `GateChangesForDeviceType`, and then registers it:

```
procedure( GateChangesForDeviceType(instId termId)
  let((net)
    when(exists(name ciGetTermNames("gate") name==termId~>name)
      when(ciIsDevice(instId "nmos")
        net = dbMakeNet(instId~>cellView "vdda_nmos")
      )
      when(ciIsDevice(instId "pmos")
        net = "vdda_pmos"
      )
    )
    net
  )
)
```

```
mgRegisterDummyNetProc('GateChangesForDeviceType)
```

mgRegisterDummyOverrideParameters

```
mgRegisterDummyOverrideParameters(
     t_libName
     t_cellNameOrRegex
     l_supplyParamsList
     [ l_signalParamsList ]
     )
     => t / nil
```

Description

Registers the parameter overrides to be applied to Modgen dummies. The order in which the override parameters are registered is the order in which the look up is processed.

Arguments

t_libName	Library containing the cellviews whose parameters are to be overwritten.
t_cellNameOrRegex	String against which cell names are to be matched. If an exact match is not found, the argument follows the SKILL rexMatchp style to check for matching strings.
l_supplyParamsList	A list of list containing name and value pairs of parameters to be overridden when the connectivity type is set to `supply`.
l_signalParamsList	A list of list containing name and value pairs of parameters to be overridden when the connectivity type is set to `signal`. If not specified, *l_supplyParamsList* is copied by default.

Value Returned

`t`	The parameter overrides were registered.
`nil`	The parameter overrides were not registered.

Examples

Registers overrides for three different libraries.

```
mgRegisterDummyOverrideParameters("libName1" "NcellName" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE")))
=> t
mgRegisterDummyOverrideParameters("libName1" "PcellName" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE") list("intParam" "3")))
=> t
mgRegisterDummyOverrideParameters("libName1" "PcellName" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE") list("intParam" "5")))
=> t
mgRegisterDummyOverrideParameters("libName1" "[A-Z]cellName" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
=> t
mgRegisterDummyOverrideParameters("libName2" "NcellName" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
```

```
=> t

mgRegisterDummyOverrideParameters("libName3" "[a-z]*cellName" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))

=> t
```

If you run `mgGetRegisteredDummyOverrideParameters` for the above example, it returns the following:

```
mgGetRegisteredDummyOverrideParameters("libName1" "PcellName" 'all)

=> (nil supply (("intParam" "3") ("boolStrParam2" "TRUE") ("boolStrParam1" "TRUE"))
    signal (("intParam" "3") ("boolStrParam2" "TRUE") ("boolStrParam1" "TRUE")))
```

mgRegisterDummyParamsRefDirection

```
mgRegisterDummyParamsRefDirection(
     t_libName
     t_cellNameOrRegex
     t_direction [ Horizontal | Vertical ]
     l_paramsList
     )
     => t / nil
```

Description

Registers Modgen dummy parameters and forcefully matches them to their nearest directional neighbor, left/right or top/bottom.

Arguments

t_libName	Library containing the cellviews.
t_cellNameOrRegex	String against which cell names are to be matched. If an exact match is not found, the argument follows the SKILL rexMatchp style to check for matching strings.
t_direction	Direction of the neighboring instance with which dummy parameters are to be matched. Valid values are `Horizontal` and `Vertical`.
l_paramsList	List of names of directional parameters.

Value Returned

`t`	The Modgen dummy directional parameters are registered.
`nil`	The Modgen dummy directional parameters were not registered.

Examples

Registers Modgen dummy directional parameters for different libraries and directions:

```
mgRegisterDummyParamsRefDirection("libName1" "NcellName" "Horizontal"
list("width" "connectGates"))
mgRegisterDummyParamsRefDirection("libName1" "NcellName" "Vertical"
list("fingers" "length"))
mgRegisterDummyParamsRefDirection("libName1" "P*" "Horizontal" list("width"
"connectSD"))
mgRegisterDummyParamsRefDirection("libName1" "P*" "Vertical" list("fingers"
"length"))
mgRegisterDummyParamsRefDirection("libName1" "N*" "Horizontal" list("width"
"connectSD"))
mgRegisterDummyParamsRefDirection("libName1" "N*" "Vertical" list("fingers"
"length"))
mgRegisterDummyParamsRefDirection("libName2" "*cellName" "Horizontal"
list("width" "connectSD"))
mgRegisterDummyParamsRefDirection("libName2" "*cellName" "Vertical"
list("fingers" "length"))
```

Retrieves information about a registered Modgen dummy directional parameters:

```
mgGetDummyRefDirectionParams("libName1" "PcellName")
=> (nil Vertical ("fingers" "length") Horizontal ("connectSD" "width"))
=> (nil Vertical ("fingers" "length") Horizontal ("connectGates" "width"))
```

Retrieves a list of DPLs for the given library name:

```
mgGetDummyRefDirectionParams("libName1")
    (("NcellName"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectGates" "width")
        ))
        ("P*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")) )
        ("N*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
    )
)
```

Retrieves a list of lists of DPLs for all registered library names:

```
mgGetDummyRefDirectionParams()
    (("libName2"
        (("*cellName"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
    )
)
("libName1"
    (("NcellName"
```

```
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectGates" "width")
        )
        )
        ("P*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )
        ("N*"
        (nil Vertical
        ("fingers" "length") Horizontal
        ("connectSD" "width")
        )
        )

    )
)
)
```

Related Topics

mgGetDummyRefDirectionParams

mgUnregisterDummyParamRefDirection

mgRegUserProc

```
mgRegUserProc(
     s_userFunction
     )
     => t / nil
```

Description

Accepts user functions based on their keywords and registers them in the Modgen code to enable callbacks.

Arguments

s_userFunction

Represents a user function symbol. Valid user functions are:

- `'mgAbutGetEnvProc`: Returns the current state of user-defined abutment types.
- `'mgAbutSetEnvProc`: Sets the current state of user-defined abutment types.
- `'mgUserShapeProc`: Registers as user shape callbacks.
- `'mgPrePlacementProc`: Is called before placement in the Modgen.
- `'mgPlacementCheckProc`: Is called after placement to either modify the constraint and rerun the placement or accept the placement and proceed with routing.
- `'mgGetIdenticalGRDefsProc`: Returns the guard ring definition, unit cell parameters, and all associated callbacks.
- `'mgRouterUserShapeProc`: Creates user-defined shapes inside the Modgen after routing.
- `'mgSurroundDefaultsProc`: Sets the default size of devices that surround adjoining Modgen instances when creating identical guard rings.

Value Returned

`t`	The specified functions were registered.
`nil`	The functions could not be registered.

Examples

Registers user functions in the Modgen code.

```
mgRegUserProc(?mgAbutGetEnvProc 'pdkGetAbutFuncName)
mgRegUserProc(?mgAbutSetEnvProc 'pdkSetAbutFuncName)
mgRegUserProc(?mgUserShapeProc 'pdkUserShapeFuncName)
mgRegUserProc(?mgPrePlacementProc 'pdkPrePlaceFuncName)
```

mgRemoveMemberFromMatchGroup

```
mgRemoveMemberFromMatchGroup(
     d_dbObjectId
     )
     => t / nil
```

Description

Removes the specified database object from its associated matchGroup.

Arguments

d_dbObjectId	The database ID of the object to be removed from its matchGroup.

Value Returned

t	The database object was removed from its matchGroup.
nil	The command was unsuccessful.

Example

Finds the trunk that is present in the second channel of a Modgen and removes it from all matchGroups of which it is a part:

```
when(window = hiGetCurrentWindow()
    when(cellView = geGetEditCellView(window)
        when(mgId=car(exists(fg cellView~>figGroups fg~>type=="modgen"))
            when(hTrunks = mgGetChannelTrunks(mgId 2 t)
            if(mgRemoveMemberFromMatchGroup(car(hTrunks)) then
            println("Object was removed from a Matched Group")
            else
            println("Object was not removed from or was not part of a matchGroup")
        )))))))
```

mgRemoveTopologyFromModgen

```
mgRemoveTopologyFromModgen(
     d_tpObjectId
     d_modgenId
     )
     => t / nil
```

Description

Removes the specified topology object from the specified Modgen.

Arguments

d_tpObjectId	A topology object ID.
d_modgenId	A Modgen figGroup, storage group, or constraint ID.

Value Returned

`t`	The topology object was removed from the specified Modgen.
`nil`	The command was unsuccessful.

Example

Returns `t` if the topology pattern is successfully removed from the Modgen that has `topo` as the topology pattern ID and `cid` as the Modgen constraint ID:

```
mgRemoveTopologyFromModgen(topo cid)
```

mgRotateLeftCB

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

Description

Rotates the selected instance to the left. To rotate an instance at least one instance should be selected.

Arguments

None

Value Returned

t	Rotates the instance to the left.
nil	The current window cellview ID is invalid.

Example

In this example, the selected instance is rotated to the left. If the orientation of the selected instance is `R90`, then it becomes `R0` after the rotate left operation is performed.

```
mgRotateLeftCB()
```

mgRotateMXCB

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

Description

Rotates the selected Modgen around its center using the MX transform.

Arguments

None

Value Returned

`t`	The Modgen is rotated.
`nil`	The Modgen is not rotated.

Example

Rotates the selected Modgen.

```
mgRotateMXCB()
```

mgRotateMYCB

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

Description

Rotates the selected Modgen around its center using the MY transform.

Arguments

None

Value Returned

`t`	The Modgen is rotated.
`nil`	The Modgen is not rotated.

Example

Rotates the selected Modgen.

```
mgRotateMYCB()
```

mgRotateR270CB

```
mgRotateR270CB(
     )
     => t / nil
```

Description

Rotates the selected Modgen around its center using the R270 transform.

Arguments

None

Value Returned

`t`	The Modgen is rotated.
`nil`	The Modgen is not rotated.

Example

Rotates the selected Modgen.

```
mgRotateR270CB()
```

mgRotateR90CB

```
mgRotateR90CB(
     )
     => t / nil
```

Description

Rotates the selected Modgen around its center using the R90 transform.

Arguments

None

Value Returned

`t`	The Modgen is rotated.
`nil`	The Modgen is not rotated.

Example

Rotates the selected Modgen.

```
mgRotateR90CB()
```

mgRotateRightCB

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

Description

Rotates the selected instance to the right. To rotate an instance at least one instance should be selected.

Arguments

None

Value Returned

`t`	Rotates the instance to the right.
`nil`	The current window cellview ID is invalid.

Example

In this example, the selected instance is rotated to the right. If the orientation of the selected instance is `R90`, then it becomes `R180` after the rotate right operation is performed.

```
mgRotateRightCB()
```

mgRouteCB

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

Description

Displays the Routing Style form for the current module.

Arguments

None

Value Returned

`t`	The Routing Style form is displayed.
`nil`	The Routing Style form is not displayed.

Example

Displays the Routing Style form.

```
mgRouteCB()
```

mgRoutePToTCB

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

Description

Routes the currently open Modgen using the Modgen pin-to-trunk router.

Arguments

None

Value Returned

`t`	The Modgen was routed.
`nil`	The Modgen was not routed.

Example

Runs the pin-to-trunk router.

```
mgRoutePToTCB()
```

mgSelectRowColCB

```
mgSelectRowColCB(
     t_type
     )
     => nil
```

Description

Selects all of the instances in the current row or column from the currently selected instance.

Argument

t_type	Selects all of the instances in the current row or column. Valid values: `row` or `col`

Value Returned

`t`	All instances in the current row or column are selected.
`nil`	Instances in the current row or column are not selected.

Examples

Selects all of the instances in the same row as the currently selected instance.

```
mgSelectRowColCB("row")
```

Selects all of the instances in the same column as the currently selected instance.

```
mgSelectRowColCB("col")
```

mgSetDummyLibCellAndParamAlias

```
mgSetDummyLibCellAndParamAlias(
     l_ActiveLibCells_and_params
     l_DummyLibCells_and_params
     )
     => t / nil
```

Description

Maps the specified library names, cell names, and parameter names. The function maps CDF parameter names for the parameters that do not have matching names, so that values can be copied from `cellA` to `cellB`.

Arguments

l_ActiveLibCells_and_params	A list consisting of pairs of active library and cell names and a list of parameter names. Example: `list(list("libNameA" "cellNameA") list("paramNameA1" "paramNameA2" ... "paramNameAN" ))`
l_DummyLibCells_and_params	A list of pairs of dummy library and cell names and a list of parameter names. Example: `list(list("libNameB" "cellNameB") list("paramNameB1" "paramNameB2" ... "paramNameBN" ))` Both arguments must contain the same number of strings in the parameter list. The parameters are mapped based on the order in the list, for example, `paramNameA1` maps to `paramNameB1` in the above example.

Value Returned

`t`	The specified library, cell, and parameters are mapped.
`nil`	The command was unsuccessful.

Example

Maps the specified libraries, cells, and parameters.

```
mgSetDummyLibCellAndParamAlias(list(list("libNameA" "cellNameA")
list("paramNameA1" "paramNameA2"))
list(list("libNameB" "cellNameB") list("paramNameB1" "paramNameB2")))
=> t
mgGetDummyLibCellAndParamAlias()
=> (list(list("libNameA" "cellNameA") list("paramNameA1" "paramNameA2"))
    list(list("libNameB" "cellNameB") list("paramNameB1" "paramNameB2")))
```

Related Topics

mgGetDummyLibCellAndParamAlias

mgSetIsLocalTrunk

```
mgSetIsLocalTrunk(
     d_trunkId
     g_isLocal
     )
     => t / nil
```

Description

Sets the specified trunk as local. Trunks inside a Modgen are called local trunks. These trunks can be connected only within the same Modgen, and not from outside the Modgen.

Arguments

d_trunkId	Database ID of the topological trunk.
g_isLocal	Boolean specifying whether the trunk is local. Valid Values: `t` if the trunk is local; `nil` if the trunk is not local.

Value Returned

`t`	The specified topological trunk was set as local or not local depending on whether *g_isLocal* is set to `t` or `nil`.
`nil`	The command was unsuccessful.

Example

Creates a topology trunk and sets it as local:

```
cv = geGetEditCellView()
net = dbFindNetByName(cv, "3")
myTopo = dbCreateTopology("myTopo" net)
myTrunk = dbCreateTrunk("myTrunk" myTopo "horizontal")

when(mgGetIsLocalTrunk(myTrunk) == nil
    mgSetIsLocalTrunk(myTrunk t)
    )
```

mgSetStrapDirection

```
mgSetStrapDirection(
     d_strapId
     t_direction
     )
```

Description

Sets the strap direction. This value is valid only for straps with a single strap object. For straps with more than one strap objects, the router determines strap direction.

Arguments

d_strapId	Database ID of the strap topology.
t_direction	Strap direction. Valid values are `horizontal` and `vertical`.

Value Returned

`t`	The strap direction was set as specified.
`nil`	The command was unsuccessful.

Example

Sets the strap direction.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap=car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
mgSetStrapDirection(strap "vertical")
```

mgSetStrapHasDirection

```
mgSetStrapHasDirection(
     d_strapId
     g_hasDirection
     )
```

Description

Specifies whether the given strap can have a direction.

Arguments

d_strapId	Database ID of the strap topology.
g_hasDirection	Boolean specifying whether the strap has a direction.

Value Returned

`t`	The specified value was set.
`nil`	The command was unsuccessful.

Example

Specifies that the strap can have a direction.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
mgSetStrapHasDirection(strap t)
```

mgSetStrapHasOffset

```
mgSetStrapHasOffset(
     d_strapId
     g_hasOffset
     )
```

Description

Specifies whether the given strap has a valid offset value.

Arguments

d_strapId	Database ID of the strap topology.
g_hasOffset	Boolean specifying whether the strap has an offset.

Value Returned

`t`	The specified value was set.
`nil`	The command was unsuccessful.

Example

Specifies that the strap cannot have an offset.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
mgSetStrapHasOffset(strap t)
mgSetStrapOffset(strap 0)
```

mgSetStrapLongOffset1

```
mgSetStrapLongOffset1(
     d_strapId
     n_longOffset1
     ) => t / nil
```

Description

Sets the long offset attribute for the bottom and left edge of the specified strap. This value determines the distance that the strap will extend from the strap object.

Arguments

d_strapId	Database ID of the strap topology.
n_longOffset1	Long offset value for the specified strap.

Value Returned

t	The specified value was set.
nil	The command was unsuccessful.

Example

Specifies the long offset value for a strap for the bottom and left edges.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
mgSetStrapLongOffset1(strap 0.5)
```

mgSetStrapLongOffset2

```
mgSetStrapLongOffset2(
     d_strapId
     n_longOffset2
     )=> t / nil
```

Description

Sets the long offset attribute for the top and right edge of the specified strap. This value determines the distance that the strap will extend from the strap object.

Arguments

d_strapId	Database ID of the strap topology.
n_longOffset2	Long offset value of the strap in user units.

Value Returned

`t`	The specified value was set.
`nil`	The command was unsuccessful.

Example

Specifies the long offset value for a strap for the top and right edges.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
mgSetStrapLongOffset2(strap 0.5)
```

mgSetStrapOffset

```
mgSetStrapOffset(
     d_strapId
     n_offset
     )
     => t / nil
```

Description

Sets the strap offset attribute, which represents the distance from the bottom or left of the bBox of the strap object.

Arguments

d_strapId	Database ID of the strap topology.
n_offset	Strap offset in user units.

Value Returned

`t`	The specified strap offset value was set.
`nil`	The command was unsuccessful.

Example

Specifies the offset attribute for a strap.

```
mgId = car(geGetEditCellView()~>figGroups)
topo = car(mgGetTopologyFromModgen(mgId))
strap = car(exists(obj dbGetTopologyObjects(topo)
dbGetTopologyPatternType(obj)=="strap"))
mgSetStrapOffset(strap 0.5)
```

mgSetTrunkRefLayerPurpose

```
mgSetTrunkRefLayerPurpose(
     d_trunkId
     t_layerName
     [ t_purposeName ]
     )
     => t / nil
```

Description

Sets the layer and purpose of a topological trunk that is anchored to an instance. The bounding box of the specified layer and purpose inside the instance is used to determine the trunk's location. In other words, the trunk's orthogonal offset is from one side of the reference layer and purpose bounding box. The purpose argument is optional. If not specified, then the purpose argument defaults to `any`. A trunk's reference layer and purpose can be removed by setting both the layer and purpose arguments to `any`.

Arguments

d_trunkId	Database ID of the topological trunk.
t_layerName	String representing the reference layer.
t_purposeName	String representing the reference purpose. This is an optional argument.

Value Returned

`t`	The layer (and purpose) of the topological trunk was set.
`nil`	The layer (and purpose) of the topological trunk could not be set.

Example

Specifies the layer and purpose for a topological trunk that is anchored to an instance.

```
cv = geGetEditCellView()
        n = dbFindNetByName(cv "net7")
        topo = car(setof(x dbGetCellViewTopologies(cv) equal(dbGetTopologyNet(x)
n)))
        trunk = car(dbGetTopologyObjects(topo))
        mgSetTrunkRefLayerPurpose(trunk "Oxide_thk" "drawing")
        ===>t
```

```
mgGetTrunkRefLayerPurpose(trunk)
===>("Oxide_thk" "drawing")
```

mgSetColumnRoutingChannelWidth

```
mgSetColumnRoutingChannelWidth(
     d_constraintId
     x_channelIndex
     f_channelWidth
     )
     => f_channelWidth / nil
```

Description

Sets the routing channel width for the specified Modgen column.

Arguments

d_constraintId	Modgen constraint ID.
x_channelIndex	Channel index indicates the Modgen column for which the specified channel width needs to be set. Column number 0 refers to channel to the left of Modgen, column number 1 indicates the first channel to the right, column number 2 indicates the second channel to the right, and so on.
f_channelWidth	Channel width to be set for the specified Modgen column.

Value Returned

f_channelWidth	The channel width that was set for the specified Modgen column.
`nil`	The channel width could not be set.

Example

The following example sets the routing channel width for the first two routing channel columns:

```
c = car(ciCacheGet(geGetEditCellView())->constraints)
when(c
    mgSetColumnRoutingChannelWidth(c 1 0.5)
    mgSetColumnRoutingChannelWidth(c 2 0.6)
    channelWidth = mgGetColumnRoutingChannelWidth(c 2)
)
```

mgSetRowRoutingChannelWidth

```
mgSetRowRoutingChannelWidth(
     d_constraintId
     x_channelIndex
     f_channelWidth
     )
     => f_channelWidth / nil
```

Description

Sets the routing channel width for the specified Modgen row.

Arguments

d_constraintId	Modgen constraint ID.
x_channelIndex	Channel index, which indicates the Modgen row for which the specified channel width needs to be set. Row number 0 refers to channel below the Modgen, row number 1 indicates the first channel, row number 2 indicates the second channel, and so on.
f_channelWidth	Channel width to be set for the specified Modgen row.

Value Returned

f_channelWidth	The channel width that was set for the specified Modgen row.
nil	The channel width could not be set.

Example

The following example sets the first two routing channel rows:

```
c = car(ciCacheGet(geGetEditCellView())->constraints)
when(c
    mgSetRowRoutingChannelWidth(c 1 0.5)
    mgSetRowRoutingChannelWidth(c 2 0.6)
    channelWidth = mgGetRowRoutingChannelWidth(c 2)
)
```

mgSwapCB

```
mgSwapCB(
     t_swap
     )
     => nil
```

Description

Swaps two selected instances, two rows of instances, or two columns of instances in Modgen.

Arguments

t_swap	Swaps the two selected instances, rows, or columns. Valid values: `inst`, `row`, or `col`.

Examples

In this example, the selected instances are swapped.

```
mgSwapCB("inst")
```

In this example, the selected rows are swapped.

```
mgSwapCB("row")
```

In this example, the selected columns are swapped.

```
mgSwapCB("col")
```

mgUnAbutCB

```
mgUnAbutCB(
     d_cellViewID
     )
     => t / nil
```

Description

Unabuts the selected instances in the current Modgen in the specified cellview.

Arguments

d_cellViewID	Database ID of the cellview.

Value Returned

t	The abutment is removed.
nil	The abutment is not removed.

Example

Unabuts the selected instances in the Modgen being modified in the specified cellview.

```
mgUnAbutCB(cv)
```

mgUnRegisterDummyNetProc

```
mgUnRegisterDummyNetProc(
     s_userProc
     )
     => t / nil
```

Description

Unregisters a function that specifies a net for a given dummy instance ID and terminal ID.

Arguments

s_userProc	Specifies the name of the function to be unregistered. The function includes two parameters—instance ID and terminal ID—and returns a `netName` or `netId`.

Value Returned

`t`	The function was unregistered.
`nil`	The command was unsuccessful.

Example

The following example first defines a procedure `GateChangesForDeviceType`, then registers it, and then unregisters the function:

```
procedure( GateChangesForDeviceType(instId termId)
  let((net)
    when(exists(name ciGetTermNames("gate") name==termId~>name)
      when(ciIsDevice(instId "nmos")
        net = dbMakeNet(instId~>cellView "vdda_nmos")
      )
      when(ciIsDevice(instId "pmos")
        net = "vdda_pmos"
      )
    )
    net
  )
)
```

```
mgRegisterDummyNetProc('GateChangesForDeviceType)
mgUnRegisterDummyNetProc('GateChangesForDeviceType)
```

mgUnregisterDummyOverrideParameters

```
mgUnregisterDummyOverrideParameters(
     [ t_libName ]
     )
     => t / nil
```

Description

Unregisters all registered Modgen dummy parameter overrides for the given library, if no library is specified, or all currently registered Modgen dummy parameter overrides.

Arguments

t_libName	Library name for which Modgen dummy parameters are to be unregistered. If not specified, all registered Modgen dummy parameter overrides are unregistered.

Value Returned

`t`	The parameter overrides were unregistered.
`nil`	The parameter overrides were not unregistered.

Examples

Unregister all overrides and then retrieves all registered overrides.

```
mgUnregisterDummyOverrideParameters()
=> t
mgGetRegisteredDummyOverrideParameters()
=> nil
```

mgUnregisterDummyParamRefDirection

```
mgUnregisterDummyParamRefDirection(
     [ t_libName ]
     [ t_cellNameOrRegex ]
     )
     => t
```

Description

Unregister either all registered Modgen dummy directional parameters for the given library name or cellNameOrRegex or all currently registered Modgen dummy directional parameters.

Arguments

t_libName	Library for which Modgen dummy directional parameters are to be unregistered. If not specified, all registered Modgen dummy directional parameters are unregistered.
t_cellNameOrRegex	Cell name or rexMatchp style for which Modgen dummy directional parameters are to be unregistered.

Value Returned

t	The Modgen dummy directional parameters were unregistered.

Examples

Unregisters Modgen dummy directional parameters for different libraries:

```
mgRegisterDummyParamsRefDirection("libName1" "NcellName" "Horizontal"
list("width" "connectGates"))
mgRegisterDummyParamsRefDirection("libName1" "NcellName" "Vertical"
list("fingers" "length"))
mgRegisterDummyParamsRefDirection("libName1" "P*" "Horizontal" list("width"
"connectSD"))
mgRegisterDummyParamsRefDirection("libName1" "P*" "Vertical" list("fingers"
"length"))
mgRegisterDummyParamsRefDirection("libName1" "N*" "Horizontal" list("width"
"connectSD"))
mgRegisterDummyParamsRefDirection("libName1" "N*" "Vertical" list("fingers"
"length"))
```

```
mgRegisterDummyParamsRefDirection("libName2" "*cellName" "Horizontal"
list("width" "connectSD"))
mgRegisterDummyParamsRefDirection("libName2" "*cellName" "Vertical"
list("fingers" "length"))
mgRegisterDummyParamsRefDirection("libName3" "*cellName" "Horizontal"
list("HorizParam1" "HorizParam2"))
mgRegisterDummyParamsRefDirection("libName3" "*cellName" "Vertical"
list("VertParam1" "VertParam2"))
```

Unregisters only the "libName1"/"N*" entry:

```
mgUnregisterDummyParamRefDirection("libName1" "N*")
mgGetDummyRefDirectionParams("libName1" "NcellName")
```

Unregisters all entries for "libName1":

```
mgUnregisterDummyParamRefDirection("libName1")
mgGetDummyRefDirectionParams()
```

Unregister all entries:

```
mgUnregisterDummyParamRefDirection()
mgGetDummyRefDirectionParams()
```

Related Topics

mgRegisterDummyParamsRefDirection

mgGetDummyRefDirectionParams

mgUnRegUserProc

```
mgUnRegUserProc(
     s_userFunction
     )
     => t / nil
```

Description

Accepts user functions based on the keywords and unregisters them in the Modgen code.

Arguments

s_userFunction

Represents a user function symbol. Valid user functions are:

- `'mgAbutGetEnvProc`: Returns the current state of user-defined abutment types.
- `'mgAbutSetEnvProc`: Sets the current state of user-defined abutment types.
- `'mgUserShapeProc`: Registers as user shape callbacks.
- `'mgPrePlacementProc`: Is called before placement in the Modgen.
- `'mgPlacementCheckProc`: Is called after placement to either modify the constraint and rerun the placement or accept the placement and proceed with routing.
- `'mgGetIdenticalGRDefsProc`: Returns the guard ring definition, unit cell parameters, and all associated callbacks.
- `'mgRouterUserShapeProc`: Creates user-defined shapes inside the Modgen after routing.
- `'mgSurroundDefaultsProc`: Sets the default size of devices that surround adjoining Modgen instances when creating identical guard rings.

Value Returned

`t`	The specified functions were unregistered.
`nil`	The functions could not be unregistered.

Examples

Unregisters the given user functions from the Modgen code.

```
mgUnRegUserProc(?mgAbutGetEnvProc 'pdkGetAbutFuncName)
mgUnRegUserProc(?mgAbutSetEnvProc 'pdkSetAbutFuncName)
mgUnRegUserProc(?mgUserShapeProc 'pdkUserShapeFuncName)
mgUnRegUserProc(?mgPrePlacementProc 'pdkPrePlaceFuncName)
```

mgUpdateCB

```
mgUpdateCB(
     S_modgenName
     d_cellViewID
     )
     => t / nil
```

Description

Lets the Modgen ConObserver update the Modgen layout module.

Arguments

s_modgenName	Name of the Modgen.
d_cellViewID	Database ID of the cellview.

Value Returned

t	The function ran successfully.
nil	The function did not run successfully.

Example

Lets the Modgen ConObserver update the Modgen module.

```
mgUpdateCB("Modgen_1" cv)
```

mgUpdateEmptyRowColumnHilightCB

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

Description

Update highlights of empty rows to follow any other changes in the Modgen.

Example

Updates visible empty row or column highlights to match any other changes to the Modgen. If empty row or column highlights are turned off, nothing happens.

```
mgUpdateEmptyRowColumnHilightCB()
```

mgUpdateHoriAlignCB

```
mgUpdateHoriAlignCB(
     d_cellViewID
     s_side
     )
     => t / nil
```

Description

Aligns the selected instances on the specified side. Side can be one of `left`, `right`, or `center`.

Arguments

d_cellViewID	Database ID of the cellview.
s_side	The side on which to align the devices.

Value Returned

`t`	The devices are aligned on the specified side.
`nil`	The devices are not aligned.

Example

Aligns the selected devices on their left sides.

```
mgUpdateHoriAlignCB(cv "left")
```

mgUpdateVertAlignCB

```
mgUpdateVertAlignCB(
     d_cellViewID
     s_side
     )
     => t / nil
```

Description

Aligns the selected instances on the specified side. Side can be one of *top*, *bottom*, or *center*.

Arguments

d_cellViewID	Database ID of the cellview.
s_side	The side on which to align the devices.

Value Returned

t	The devices are aligned on the specified side.
nil	The devices are not aligned.

Example

Aligns the selected devices on their top edges.

```
mgUpdateVertAlignCB(cv "top")
```

gpeAddInstance

```
gpeAddInstance(
     d_modgenID
     l_instances
     )
     => t / nil
```

Description

Adds the given list of instances to the specified Modgen.

Arguments

d_modgenID	Modgen figGroupId to which instances must be added.
l_instances	List of schematic or layout instances to be added to the Modgen.

Value Returned

t	The instances were added to the Modgen.
nil	The command was unsuccessful.

Example

The following procedure first identifies a figGroup in the cellview `cv`, and then assigns it to the specified Modgen `modgenid`. Instances in the cellview are identified and assigned to `inst1` and `inst2`. These two instances are then added to `modgenid`.

```
cv=geGetEditCellView()
modgenid=leGetEditFigGroup()
inst1=dbFindAnyInstByName(cv "M1")
inst2=dbFindAnyInstByName(cv "M2")
gpeAddInstance(modgenid list(inst1 inst2))
```

gpeClearPresetGenerators

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

Description

Deletes all the registered preset generator functions from the system.

Arguments

None

Value Returned

`t`	All the registered preset generator functions were deleted.
`nil`	The command was unsuccessful.

Example

The following procedure first defines a preset function. Custom checks are performed to determine when the preset function has to be displayed. Then the function is registered and tested. Here, the return value is `t`.

Then, all preset functions are deleted from the system and the function is tested for registry. Now the return value is `nil`.

```
procedure(test_preset(sbox @optional selection) print("Hello world"))
=> test_preset
gpeUnregisterPresetGen("test_preset")
=> t if registered; nil if not registered
gpeRegisterPresetGen("gen1" "test_preset" "")
=> t
gpeIsRegisteredPresetGen("test_preset")
=>t
gpeClearPresetGenerators()
=> t
gpeIsRegisteredPresetGen("test_preset")
=> nil
```

gpeExtractReuseTemplate

```
gpeExtractReuseTemplate(
     u_constraint | d_fig | l_instIDs
     g_structureName
     t_fileName
     )
     => t | nil
```

Description

Extracts the given Modgen into a reusable text file.

Arguments

u_constraint | d_fig | l_instIDs

Source Modgen constraint ID, figGoup ID, or list of instances.

g_structureName

Category to which the Modgen belongs.

t_fileName

Name of the text file to which the Modgen must be saved.

Value Returned

`t`	The given Modgen was extracted into a text file.
`nil`	The given Modgen could not be extracted into a text file.

Example

Extract the Modgen `fg` of type `diffPair` into a text file `pair1.txt`. The file is saved at location `./.cadence/dfII/modgens/templateLib/DiffPair/pair1.txt`.

```
fg = car(exists(x geGetSelectedSet() (x~>objType=="figGroup" &&
x~>type=="modgen")))
gpeExtractReuseTemplate(fg "DiffPair" "pair1.txt")
```

gpeExtractReuseTemplatesFromLCV

```
gpeExtractReuseTemplatesFromLCV(
     l_libNames
     [ l_cellNames ]
     [ l_viewNames ]
     [ g_includeNonUserDefinedGroups ]
     )
     => t | nil
```

Description

Extracts array reuse template files for recognized groups and arrays for all combinations of the specified libraries, cells, and views in the current design.

Arguments

l_libNames

Names of libraries from which reuse templates are to be extracted.

l_cellNames

Names of cells from which reuse templates are to be extracted. When unspecified, all cells in the specified libraries are considered.

l_viewNames

Names of views from which reuse templates are to be extracted. When unspecified, all views in the specified libraries and cells are considered.

g_includeNonUserDefinedGroups

Specifies whether templates can be extracted from non-user defined groups and flat layouts. The default value is `nil`.

Value Returned

`t`	All groups or arrays in the specified set of source designs have been extracted.
`nil`	Either all possible groups in the specified source designs failed or an error occurred to prevent the function from running to completion, for example, insufficient licensing.

Examples

Extracts templates for all the recognized groups or defined arrays for the combination of the specified library, cell, and view.

```
gpeExtractReuseTemplatesFromLCV("lib1" "cell1" "view1")
=> t
```

Extracts templates for all the recognized groups or defined arrays under the specified library. Not specifying the optional arguments cell and view names results in considering all cells and views in the specified library.

```
gpeExtractReuseTemplatesFromLCV("lib2")
=> t
```

gpeExtractTemplateFromMG

```
gpeExtractTemplateFromMG(
     u_constraintCache
     )
     => t / nil
```

Description

Generates a reusable template for each Modgen constraint present in the specified constraint cache.

Arguments

u_constraintCache

Name of the constraint cache.

Value Returned

`t`	All Modgens in the given constraint cache have been extracted into reusable templates.
`nil`	The command was unsuccessful.

Example

Generates reusable templates for all Modgens in the current cellview:

```
cv = geGetEditCellView()
cache = ciCacheGet(cv)
gpeExtractTemplateFromMG(cache)
=> t
```

gpeFindArrayPatternDiffs

```
gpeFindArrayPatternDiffs(
     g_ref1
     g_ref2
     )
     => t / nil
```

Description

Compares array patterns of the specified Modgen constraints or sandbox objects and reports differences, if any.

Arguments

g_ref1	Specifies a schematic or layout constraint or a sandbox object.
g_ref2	Specifies another schematic or layout constraint or sandbox object.

Value Returned

`t`	Both arguments are valid.
`nil`	One or more arguments are invalid.

Messages	In addition to the above return values, messages stating whether the size and pattern of the two arrays match are displayed.
	Differences in array patterns are reported in the following format in the message:
	`Row=1 Column=2 M0(schematic) M1 (layout)`
	`Row=N Column=M MX(schematic) MY (layout)`

Examples

Compares Modgens `scon` and `lcon` present in the schematic and layout views and reports differences, if any.

```
gpeFindArrayPatternDiffs(scon lcon)
```

Compares GPE sandboxes `sbox1` and `sbox2` and reports differences, if any.

```
gpeFindArrayPatternDiffs(sbox1 sbox2)
```

gpeGetGridValue

```
gpeGetGridValue(
     g_tableView
     [ g_location ]
     [ s_type ]
     )
     => grid_value
```

Description

Returns the value of the cells in the Grid Pattern Editor table view.

Arguments

g_tableView	The tableView of the grid.
g_location	Location of cells. Valid values are: ■ Unspecified: The current selection is used. ■ `t`: All cells are returned. ■ A specific `row:col`: Its cell value is returned. ■ `list(row1:col1 row2:col2)`: A list of the two cell values is returned. ■ `list(list(bottom:left top:right) ...)`: A list of cell values representing the specified range is returned.
s_type	A symbol indicating the type of value to retrieve. If unspecified, then the current display value is used. Valid values are `'symbol`, `'layoutName`, `'schematicName`, `'orient`, and `'display` (default).

Value Returned

grid_value	The value of the grid at the specified locations.

Examples

Returns the displayed value of the grid cells currently selected:

```
getGridValue(dwindow('gpeGridAsst_1)->tableView)
```

Returns the displayed values of all the cells in the grid, arranged as a list of one list per row:

```
getGridValue(dwindow('gpeGridAsst_1)->tableView t)
```

Returns the orientation of the cell at row 1, column 2:

```
getGridValue(dwindow('gpeGridAsst_1)->tableView 1:2 'orient)
```

Returns the orientation of all cells that have the symbol A:

```
getGridValue(dwindow('gpeGridAsst_1)->tableView '('symbol "A") 'orient)
```

gpeGetSelection

```
gpeGetSelection(
     g_tableView
     )
     => list(row column) / list(bottom:left top:right)
```

Description

Returns a list of currently selected cells or a range of cells in a grid-pattern-editor table format.

Arguments

g_tableView	The tableView object.

Value Returned

list(*row column*)	The return value for a single cell. This format can also be used for multiple cells.
list(*bottom*:*left top*:*right*)	An alternate return value format for multiple cells, depending on the values retrieved.

Examples

To check if any instances are selected in the Grid Pattern Mapping assistant:

```
not(null(gpeGetSelection(dwindow('gpeMappingAsst_1)->tableView)))
```

Assume a 4x4 grid, with all `M1` cells in the first column. To find all `M1` cells in the Grid Pattern Editor assistant:

```
tv = dwindow('gpeGridAsst_1)->tableView
gpeSelect(tv '(schematicName "M1"))
gpeGetSelection(tv)
=>((0 0) (1 0) (2 0) (3 0))
```

gpeIsPresetGenDisplayable

```
gpeIsPresetGenDisplayable(
     t_functionName
     d_figGroupId
     )
     => t / nil
```

Description

Checks whether the given preset generator is displayed in the *Preset* drop-down of the Grid Pattern Editor for the selected Modgen.

Arguments

t_functionName	Specifies the name of the function that needs to be checked for display.
d_figGroupId	Specifies the database ID of the Modgen figGroup to be checked for the specified preset display.

Value Returned

t	The preset generator is displayed in the *Preset* drop-down list of the Grid Pattern Editor.
nil	The function is not displayed in the *Preset* drop-down list of the Grid Pattern Editor.

Example

The following procedure first defines a preset function. Custom checks are performed to determine when the preset function is displayed. Then the function is registered and tested and whether the given preset generator can be displayed in the *Preset* drop-down list of the Grid Pattern Editor.

```
procedure(test_preset(sbox @optional selection) print("Hello world"))
=> test_preset
gpeUnregisterPresetGen("test_preset")
=> t if registered; nil if not registered
procedure(test_preset_check(args) nil)
=> test_preset_check
```

```
gpeRegisterPresetGen("gen1" "test_preset" "test_preset_check")
=> t
gpeIsRegisteredPresetGen("test_preset")
=>t
a = car(setof(fg geGetEditCellView()~>figGroups fg~>type=="modgen"))
=> db:0x649b1c9a
gpeIsPresetGenDisplayable("test_preset" a)
=> nil
```

gpeIsRegisteredPresetGen

```
gpeIsRegisteredPresetGen(
     t_functionName
     )
     => t / nil
```

Description

Checks whether the specified function is a registered preset generator function.

Arguments

t_functionName	Name of the function that needs to be checked for registry.

Value Returned

`t`	The function is a registered preset generator function.
`nil`	The function is not a registered preset generator function.

Example

The following procedure first defines a preset generator function. Custom checks are performed to determine when the preset function has to be displayed. There are two tests performed. The first one after registration, where the return value is `t`. The second one is when the function is unregistered and tested. Here, the return value is `nil`.

```
procedure(test_preset(sbox @optional selection) print("Hello world"))
=> test_preset
gpeUnregisterPresetGen("test_preset")
=> t if registered; nil if not registered
gpeRegisterPresetGen("gen1" "test_preset" "")
=> t
gpeIsRegisteredPresetGen("test_preset")
=>t
gpeClearPresetGenerators()
=> t
gpeIsRegisteredPresetGen("test_preset")
=> nil
```

gpeLoadReuseTemplate

```
gpeLoadReuseTemplate(
     l_target
     t_structureName
     t_fileName
     [ l_mapping ]
     [ n_rows ]
     )
     => u_sandbox / nil
```

Description

Creates a new Modgen constraint by applying the given Modgen reuse template to the specified list of instances.

Arguments

l_target	Lists the instance IDs, Modgen IDs, or figroup IDs to which the Modgen reuse template is to be applied.
t_structureName	Specifies the category directory that contains the Modgen reuse template.
t_fileName	Specifies the Modgen template file name to be applied.
l_mapping	Lists pairs of instance names and symbol names to be mapped.
n_rows	Specifies the number of rows to be generated in the new Modgen.

Value Returned

u_sandbox	ID of the Modgen sandbox object that was created.
`nil`	The command was unsuccessful.

Example

Applies the `dfII.txt` template to a set of instances to create a new Modgen.

```
inst1 = dbFindAnyInstByName(geGetEditCellView() "M11")
inst2 = dbFindAnyInstByName(geGetEditCellView() "M12")
gpeLoadReuseTemplate(list(inst1 inst2) "MOS_Differential_Pair" "diff1.txt")
```

The following example uses optional arguments to specify mapping and number of rows in the Modgen.

```
gpeLoadReuseTemplate(list(inst1 inst2) "MOS_Differential_Pair" "diff1.txt"
?mapping list(list(inst1~>name "A") list(inst2~>name "B")) ?rows 2)
```

gpeMove

```
gpeMove(
     g_tableView
     [ g_cells ]
     g_fromAnchor
     g_toAnchor
     )
     => t / nil
```

Description

Moves a list of cells within the Grid Pattern Editor table from one point to another.

Arguments

g_tableView	The tableView for which cells need to be moved.
g_cells	(Optional) List of cells to move. This option defaults to the current selection.
g_fromAnchor	A cell, usually in the list of cells, to be moved.
g_toAnchor	The target cell.

Value Returned

t	The specified list of cells have been moved within the Grid Pattern Editor table.
nil	The cells could not be moved.s

Examples

In a 4x4 grid, to move a cell from the top right corner to the bottom left corner:

```
tv = dwindow('gpeGridAsst_1)->tableView  ; get tableView
gpeSelect(tv 3:3)      ; select top right corner cell
gpeMove(tv 3:3 0:0)    ; move from top right to bottom left
```

To move three cells in the top left corner down by two rows:

```
gpeMove(tv list(3:0 3:1 2:0) 2:0 0:0)
```

gpePresetReorientResistor

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

Description

Flips the orientations of the members of a preselected Modgen to reduce the overall connectivity wire length. The placement of the members is not altered.

Arguments

None

Value Returned

`t`	The orientations of the Modgen members were flipped.
`nil`	The command was unsuccessful.

Example

Select a Modgen in the design canvas or in the Constraint Manager. Call the following command to flip the orientations of the Modgen members. Check the trace lines to view changes.

```
gpePresetReorientResistor()
```

gpeRegisterPresetGen

```
gpeRegisterPresetGen(
     t_displayName
     t_functionName
     t_checkFunctionName
     [ g_noStacks ]
     )
     => t / nil
```

Description

Registers the specified function as a preset generator.

Arguments

t_displayName	Display name of the preset generator. This name is displayed in the *Preset* drop-down list of the Grid Pattern Editor.
t_functionName	The generator function name.
t_checkFunctionName	A custom function used to pre-process the selected target, which is the Modgen sandbox. The function returns values `t` or `nil`, which can be used to specify whether the preset is to be displayed in the *Preset* drop-down. To ensure that the preset is always displayed in the drop-down, set *checkFunctionName* to `""`. Alternatively, you can use `procedure(test_preset_check(args) t)`, but this method is not recommended as it adversely impacts the tool performance.
g_noStacks	Specifies whether the sandbox must be in the unstacked format. If this argument is set to nil or is omitted, the sandbox that is provided to the preset callback is in the stacked format.

Value Returned

`t`	The specified function has been registered.
`nil`	The command was unsuccessful.

Example

A preset function is defined and custom checks are performed to determine when the preset function has to be displayed. The preset function is then registered.

```
procedure(test_preset(sbox @optional selection) print("Hello world"))
=> test_preset
gpeUnregisterPresetGen("test_preset")
=> t if registered; nil if not registered
gpeRegisterPresetGen("gen1" "test_preset" "")
=> t
```

gpeRemoveInstance

```
gpeRemoveInstance(
     d_modgenID
     l_instances
     )
     => t / nil
```

Description

Removes the given instances from the specified Modgen figGroup.

Arguments

d_modgenID	Modgen figGroup ID from which instances must be removed.
l_instances	List of schematic or layout instances to be removed from the Modgen.

Value Returned

t	The instances were removed from the Modgen.
nil	The command was unsuccessful.

Example

The following procedure first identifies a figGroup in the cellview `cv` and assigns it to the specified Modgen `modgenid`. Instances in the cellview are identified and assigned to `inst1` and `inst2`. These two instances are then removed from `modgenid`.

```
cv = geGetEditCellView()
modgenid = leGetEditFigGroup()
inst1 = dbFindAnyInstByName(cv "M1")
inst2 = dbFindAnyInstByName(cv "M2")
gpeRemoveInstance(modgenid list(inst1 inst2))
```

gpeRunPresetGen

```
gpeRunPresetGen(
     t_functionName
     d_figGroupId
     )
     => t / nil
```

Description

Invokes the specified preset generator function on the active figGroup. The figGroup is updated according to the preset generator logic. Ensure that the Grid Pattern Editor (GPE) assistant is open when running this command. Also ensure that the selection argument specifies a valid Modgen figGroup and that the preset is registered correctly.

Arguments

t_functionName	Name of the preset generator function to be invoked.
d_figGroupId	Database ID of the currently active figure group.

Value Returned

t	The preset generator function was invoked.
nil	The preset generator function could not be invoked.

Example

In the following example, first the preset function is defined. Custom checks are performed on the preset function to determine when the preset function has to be displayed. Finally, the preset function is registered.

```
procedure(test_preset(sbox @optional selection) print("Hello world"))
=> test_preset
gpeUnregisterPresetGen("test_preset")
=> t
gpeRegisterPresetGen("gen1" "test_preset" "")
=> t
gpeRunPresetGen("gen1" css())
=> Hello world
=> nil
```

gpeSelect

```
gpeSelect(
     g_tableView
     [g_selectionList | g_type g_value]
     )
     => t / nil
```

Description

Select cells in the Grid Pattern Editor or Grid Pattern Mapping assistants.

Arguments

g_tableView	The tableView in which all cell items need to be selected.
g_selectionList | *g_type g_value*	*g_selectionList* is a list or range of cells to be selected, or a list of type and values to search for. If no types are specified, then the display value is searched. Set the value to t to select all rows, or nil to clear the selection. *g_type* is a symbol indicating the type of value to search for. *g_value* is the value to search for.

Value Returned

t	The cells were selected.
nil	The cells could not be selected.

Examples

Assuming a 4x4 grid with A, B, C, D symbols in the columns, and rows alternating between R0 and MY orientation.

```
tv = dwindow('gpeGridAsst_1)->tableView
gpeSelect(tv t) ; select all by row
gpeGetSelection(tv)
--> (((3 0) (3 3)) ((2 0) (2 3)) ((1 0) (1 3)) ((0 0) (0 3)))

gpeSelect(tv nil) ; select none
gpeGetSelection(tv)
--> nil
```

```
gpeSelect(tv list(list(0:0 3:0))) ; select left-most column
gpeGetSelection(tv)
--> (((0 0) (3 0)))

gpeSelect(tv '("B")) ; select all the B cells
gpeGetSelection(tv)
--> ((0 1) (1 1) (2 1) (3 1))

gpeSelect(tv '(orient "MY" symbol "C")) ; select all the C cells with MY orientation
gpeGetSelection(tv)
--> ((1 2) (3 2))
```

gpeSetGridText

```
gpeSetGridText(
     g_tableView
     t_pattern
     )
     => t / nil
```

Description

Specifies a textual grid pattern for the *Pattern* field on the *Text* tab of the Grid Pattern Editor assistant.

Arguments

g_tableView	Specifies the `tableView` of the grid.
t_pattern	Specifies a textual grid pattern.

Value Returned

`t`	The textual pattern was assigned.
`nil`	The command was unsuccessful.

Examples

Specifies a textual grid pattern `G H\nE F\nC D\nA B` to be assigned to the grid. New lines are indicated by `\n` and empty spaces are used for separating symbols. The pattern is read from left to right, which corresponds to top left to right bottom in the grid:

```
gpeSetGridText(dwindow('gpeGridAsst_1)->tableView "G H\nE F\nC D\nA B")

dwindow('gpeGridAsst_1) => docked window corresponding to GUI gpe grid pattern
dwindow('gpeGridAsst_1)->tableView  => text pattern widget id for that GUI
```

gpeSetGridValue

```
gpeSetGridValue(
     g_tableView
     [ g_location ]
     S_value
     [ s_type ]
     )
     => t / nil
```

Description

Sets the value of the Grid Pattern Editor table view cells. If there is a single cell selected before the call, the next cell will be selected automatically.

Arguments

g_tableView	Name of the tableView
g_location	Location of cells. ■ If unspecified, then the current selection will be used. ■ If set to t, returns all cells. ■ If set to row:col, returns the cell value at (row,col). ■ If set to list(row1:col1 row2:col2), returns a list of the two cell values. ■ If set to list(list(bottom:left top:right) ...), returns a list of cell values representing the specified ranges.
s_value	Value as a string
s_type	A symbol indicating the type of value to retrieve; if unspecified, then the current display value is used. Valid values are `'symbol`, `'layoutName`, `'schematicName`, `'orient`, or `'display` (default).

Value Returned

`t`	The value of the Grid Pattern Editor table view cells was set.
`nil`	The value of the Grid Pattern Editor table view cells could not be set.

Example

Sets the current selected cells to a value of `A`:

```
getSridValue(dwindow('gpeGridAsst_1)->tableView "A")
```

Clears all cells in the grid or table and sets them all to " ":

```
setGridValue(dwindow('gpeGridAsst_1)->tableView t " ")
```

Sets the cell orientation at row 1, column 2 to `R90`. Returns `nil` if row 1, column 2 is empty:

```
setGridValue(dwindow('gpeGridAsst_1)->tableView 1:2 "R90" 'orient)
```

Sets the orientation of all cells with symbol `A` to mirror-on-X-axis (`MX`):

```
setGridValue(dwindow('gpeGridAsst_1)->tableView '(symbol "A") "MX" 'orient)
```

gpeSetOrientText

```
gpeSetOrientText(
     g_tableView
     t_orientation
     )
     => t / nil
```

Description

Specifies the grid pattern orientation for the *Orient* field on the *Text* tab of the Grid Pattern Editor assistant.

Arguments

g_tableView	Specifies the `tableView` of the grid.
`?orientation` *t_orientation*	Specifies the grid pattern orientation.

Value Returned

`t`	The grid pattern orientation was assigned.
`nil`	The command was unsuccessful.

Examples

Specifies orientation `(R0)2\n(R0)2\n(R0)2\n(R0)2` to be set to the grid. New lines are indicated by `\n` and empty spaces are used for separating symbols. The orientation assignment is created by reading from left to right, which corresponds to top left to right bottom in the grid:

```
gpeSetOrientText(dwindow('gpeGridAsst_1)->tableView "(R0)2\n(R0)2\n(R0)2\n(R0)2")
dwindow('gpeGridAsst_1) => docked window corresponding to GUI gpe grid pattern
dwindow('gpeGridAsst_1)->tableView  => orient pattern widget id for that GUI
```

For this assignment, `(R0)2\n(R0)2\n(R0)2\n(R0)2` expands to:

```
"R0 R0
 R0 R0
 R0 R0
 R0 R0"
```

```
gpeSetOrientText(dwindow('gpeGridAsst_1)->tableView "R0 MX\n R0 R180\nR90 MY\nR0
R0")
```

For this assignment, `R0 MX\n R0 R180\nR90 MY\nR0 R0` expands to:

```
"R0  MX
 R0  R180
 R90 MY
 R0 R0"
```

gpeSetSize

```
gpeSetSize(
     g_tableView
     l_dimensions
     ['stable | 'nostable]
     [ n_rows ]
     [ n_columns ]
     )
     => t / nil
```

Description

Resizes the specified Grid Pattern Editor table. If the table dimensions are smaller or larger than the new dimensions, then rows or columns are added or removed, as needed.

Arguments

g_tableView	The tableView name.
l_dimensions	New dimensions of the Grid Pattern Editor table.
'stable | 'nostable	Stability status of the table.
n_rows	Number of rows to be generated in the new table.
n_columns	Number of columns to be generated in the new table.

Value Returned

t	The Grid Pattern Editor table could be resized.
nil	The Grid Pattern Editor table could not be resized.

Example

Resizes the current table to 4 rows by 4 columns:

```
gpeSetSize(dwindow('gpeGridAsst_1) 4 4)
```

Resizes a 4x4 grid to 8x8, while rearranges the 16 original cells to the bottom 2 rows (2 rows x 8 columns) to the bottom 2 rows (2 rows x 8 columns):

```
gpeSetSize(dwindow('gpeGridAsst_1) 8:8 'nostable)
```

gpeUnregisterPresetGen

```
gpeUnregisterPresetGen(
     t_functionName
     )
     => t / nil
```

Description

Unregisters the given preset generator function from the Grid Pattern Editor.

Arguments

t_functionName	Name of the preset generator function to be unregistered.

Value Returned

t	The preset generator function was unregistered.
nil	The preset generator function could not be unregistered.

Example

The following procedure first defines the preset generator function. Custom checks are performed to determine when the preset function has to be displayed. Then the function is registered and tested. Next, the function is unregistered and tested.

```
procedure(test_preset(sbox @optional selection) print("Hello world"))
=> test_preset
gpeUnregisterPresetGen("test_preset")
=> t if registered; nil if not registered
gpeRegisterPresetGen("gen1" "test_preset" "")
=> t
gpeIsRegisteredPresetGen("test_preset")
=>t
gpeUnregisterPresetGen("test_preset")
=> t
gpeIsRegisteredPresetGen("test_preset")
=> nil
```

gpeUpdateInstanceFromSchematic

```
gpeUpdateInstanceFromSchematic(
     d_modgenID
     )
     => t / nil
```

Description

Updates the given Modgen instance parameters based on the binding schematic cellview.

Arguments

d_modgenID	Modgen figGroup ID to be updated.

Value Returned

t	The Modgen instance parameters were updated.
nil	The command was unsuccessful.

Example

Updates the parameters for the given Modgen from its binding schematic cellview.

```
cv=geGetEditCellView()
modgenid=leGetEditFigGroup()
gpeUpdateInstanceFromSchematic(modgenid)
```

Modgen Placement and Routing Functions

A Modgen is stored in the form of a Modgen Pattern Editor User Type Description Object, also referred to as a Modgen sandbox object. You can use various Modgen placement and routing SKILL functions to create and place Modgen placement and routing objects without exposing the full complexity of the Modgens. Modgen sandbox objects can be created and edited from both, schematic and layout cellviews.

There are four main sections in a Modgen sandbox object: map, grid, abut, and topology. Each section contains properties and additional disembodied property lists (DPLs) that hold information about the Modgen.

They are created and added to the top-level section using specific APIs. Once created, these sections can also be used as template parameters to create additional DPLs.

Example of a Modgen Sandbox Object:

```
pe = gpeStartSandbox( mg )
     gpeGetMap( pe )
     gpeGetGrid( pe )
     gpeGetAbut( pe )
     gpeGetTrunkChains( pe )
     gpeFinishSandbox (mg)
```

Tip

A disembodied property list (DPL) is logically equivalent to a record in which new fields can be dynamically added or removed. You can use the arrow operator (->) to store and retrieve properties in the list. A DPL starts with a SKILL data object, usually nil, followed by alternating name-value pairs.

It is recommended that all the GPE place and route scripts start with a call to `gpeStartSandbox`, and finish with a call to `gpeFinishSandbox`. To discard the edits from the script, use `gpeDeleteSandbox`.

These APIs support an optional `?sync` argument. Either set this argument to `t` or use the `gpeSyncSandbox` API to turn on the synchronization mode. As a result, changes made to a Modgen sandbox object are automatically updated to the associated Modgen constraint and the layout placement.

The handlers returned by the `gpeStartSandbox, gpeEditSandbox`, and `gpeCreateSandbox` are valid until:

- The `gpeFinishSandbox` API is called.
- A different Modgen is selected using the Modgen on-canvas commands.
- The Modgen is deleted.
- The session is terminated.
- The cellview is closed.
- The Modgen `ciCon` is deleted.

All functions have a `?verbose` argument, which is set to `t` by default.

Related Topics

Disembodied Property Lists

gpeStartSandbox

gpeEditSandbox

gpeCreateSandbox

gpeDeleteSandbox

gpeFinishSandbox

gpeSyncSandbox

gpeGetMap

gpeGetGrid

gpeGetAbut

gpeAbutGridEntries

```
gpeAbutGridEntries(
     u_sandbox
     l_gridEntries
     [ ?abutType { x_abutType | t_abutTypeName } ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Abuts the specified list of instances or indexes using the given abutment type in the given sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
l_gridEntries	Includes a list of entries that are specified by indexes. Example: '('(1 1) '(1 2) ...)
[?abutType { *x_abutType* | *t_abutTypeName* }]	Specifies the abutment type, which is either a number or the abutment type name. If not specified, the default abutment type is used.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The instances were abutted.
nil	The command was unsuccessful.

Example

Makes the Modgen sandbox `sb` editable and turns on the synchronization mode. `gpeAbutGridEntires` is then used to abut a specified list of instances.

```
> sb = gpeEditSandbox()
> gpeAbutGridEntries(sb list(list(1 0) list(1 1)) ?abutType "type0" ?verbose t)
> gpeSyncSandbox(sb)
```

gpeAddDummy

```
gpeAddDummy(
     u_sandbox
     t_direction
     [ ?n x_rowsCols ]
     [ ?dummyConfig l_dummyConfig ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Adds the specified number of dummy rows or columns in the given direction to the specified Modgen sandbox object.

The dummy configuration entry, if specified, is used to configure the default entries. The master for the generated dummy entries is set to either the dummy configuration entry, which provides the master lib:cell:view specification, or the grid entries. Even if the dummy configuration entry specifies the source, the value is ignored and source is calculated based on the grid entries.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
t_direction	Indicates the side of the sandbox grid to which dummy rows or columns must be added. Valid values are: `top`, `bottom`, `left`, and `right`.
?n *x_rowCols*	Specifies the number of dummy rows or columns to be added.
?dummyConfig *l_dummyConfig*	Specifies a DPL that indicates how the dummy entries must be configured.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The dummy rows or columns were added.
nil	The command was unsuccessful.

Example

Defines a Modgen sandbox `sbox`, and then adds dummy rows to the left side of the sandbox:

```
sbox = gpeEditSandbox()
gpeAddDummy(sbox "left")
=> t
```

gpeAddDummyInEmptyCells

```
gpeAddDummyInEmptyCells(
     u_sandbox
     [ ?dummyConfig l_dummyConfig ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Fills the empty cells in the specified sandbox object with dummies.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?dummyConfig *l_dummyConfig*	Specifies a DPL that indicates how the dummy entries must be configured.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	Dummies were added to the specified sandbox object.
nil	The command was unsuccessful.

Example

Defines a Modgen sandbox `sbox`, and then adds dummies in the empty cells:

```
sbox = gpeEditSandbox()
gpeAddDummyInEmptyCells(sbox)
=> t
```

gpeAddDummySurround

```
gpeAddDummySurround(
     u_sandbox
     [ ?dummyConfig l_dummyConfig ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Adds a ring of dummies around the specified sandbox grid.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?dummyConfig *l_dummyConfig*	Specifies a DPL that indicates how the dummy entries must be configured.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The surround dummies were added.
nil	The command was unsuccessful.

Example

Defines a Modgen sandbox `sbox`, and then adds dummy rows to the left side of the sandbox:

```
sbox = gpeEditSandbox()
gpeAddDummySurround(sbox)
gpeSyncSandbox()
```

gpeCancelSandbox

```
gpeCancelSandbox(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Discards the specified Modgen sandbox object. Changes made to the Modgen sandbox object are lost. This function can be run in both schematic and layout cellviews. If run from the schematic view, the associated scratch cellview is also deleted. The Modgen constraint is not transferred to the schematic view and the dummies are not backannotated.

Arguments

u_sandbox	Specifies the Modgen sandbox object to be discarded.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The Modgen sandbox object was discarded.
nil	The Modgen sandbox object could not be discarded.

Example

In the following example, a Modgen sandbox is created and then discarded:

```
sbox = gpeStartSandbox()
pe:0x346746e0
gpeCancelSandbox(sbox)
t
sbox
pe:(nil)
```

gpeClearGridEntries

```
gpeClearGridEntries(
     u_sandbox
     [ ?row x_row ]
     [ ?col x_col ]
     [ ?verbose { t | nil } ]
     )
     => l_gridEntries / nil
```

Description

Unplaces the grid entries that match the given identifiers in the given sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?row *x_row*	Specifies the grid entry row index.
?col *x_col*	Specifies the grid entry column index.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_gridEntries	Returns a list of grid entries that were unplaced.
nil	The command was unsuccessful.

Example

Clears the grid entries in the specified rows and columns:

```
sb = gpeEditSandbox()
ge = gpeClearGridEntries(sb ?row 0 ?col 1)
gpeClearGridEntries(sb ?row 1)
gpeClearGridEntries(sb ?col 1)
```

gpeCopyColAbutment

```
gpeCopyColAbutment(
     u_sandbox
     x_sourceCol
     x_targetCol
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Copies abutment information from one column to another.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
x_sourceCol	Specifies the source column from which abutment information must be copied. Valid values are from 0 through the number of columns in the Modgen.
x_targetCol	Specifies the target column to which abutment information must be copied. Valid values are from 0 through the number of columns in the Modgen.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Abutment information was copied from the source to the target column.
`nil`	The command was unsuccessful.

Example

Copies abutment information from column 1 to column 2 in the given Modgen sandbox object:

```
sbox = gpeEditSandbox()
==> pe:#####
gpeCopyColAbutment(sbox 1 2 t)
==> t
```

gpeCopyRowAbutment

```
gpeCopyRowAbutment(
     u_sandbox
     x_sourceRow
     x_targetRow
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Copies abutment information from one row to another.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
x_sourceRow	Specifies the source row from which abutment information must be copied. Valid values are from 0 through the number of rows in the Modgen.
x_targetRow	Specifies the target row to which abutment information must be copied. Valid values are from 0 through the number of rows in the Modgen.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Abutment information was copied from the source to the target column.
`nil`	The command was unsuccessful.

Example

Copies abutment information from column 1 to column 2 in the given Modgen sandbox object:

```
sbox = gpeEditSandbox()
==> pe:#####
gpeCopyColAbutment(sbox 1 2 t)
==> t
```

gpeCreateAbutEntries

```
gpeCreateAbutEntries(
     [ ?rows l_rows ]
     [ ?cols l_columns ]
     [ ?abutType l_abutType ]
     [ ?template l_template ]
     [ ?verbose { t | nil } ]
     )
     => l_abutEntry / nil
```

Description

Creates a list of abut entries that are in line with the information in the arguments.

Arguments

`?rows` *`l_rows`*

Specifies the rows to be abutted. Valid formats are:

- An integer representing a specific row.
- A list or a range of rows. Each entry in the list must either be an integer or a list of one or two integers.

Examples:

`rows = list(1 3 5)` ==> rows 1, 3, and 5 are abutted.

`rows = list( list(0 4) )` ==> rows 0, 1, 2, 3, and 4 are abutted.

`rows = list( list(5) )` ==> row 5 is abutted.

`rows = list(0 list(1 3) list(4) )` => rows 0, 1, 2, 3, 4 and 5 are abutted.

If not specified, all rows are abutted.

`?cols` *`l_columns`*

Specifies the columns to be abutted. It is a list of lists of integers that specify the range (start end) of the columns to be abutted. In the following example, columns 0 to 2 and columns 5 to 7 are abutted:

```
cols = list( list(0 2) list(5 7) )
```

If not specified, all columns are abutted.

`?abutType` *`l_abutType`*

Specifies how abutment must be performed. You typically create the `abutType` using the `gpeCreateAbutType` API. Valid values are either a DPL or a string that specifies an abutment type that is present in the placement. If an invalid abutment type string is provided, the default type is used.

`?template` *`l_template`*

Specifies a Modgen abutment entry, which serves as a template.

`?verbose` { `t` | `nil` } Controls the display of warning messages.

Value Returned

`l_abutEntry`	A list of abutment entries. Each abutment entry is a DPL.
`nil`	Could not create a Modgen sandbox abutment entry.

Example

Creates a Modgen sandbox abutment entry with the specified arguments:

```
gpeCreateAbutEntries(?rows list(0:6) ?cols list(list(0 6)))
```

gpeCreateAbutType

```
gpeCreateAbutType(
     [ ?syncChains { t | nil } ]
     [ ?preserveRows { t | nil } ]
     [ ?permute { t | nil } ]
     [ ?mirrorEquivOrients { t | nil } ]
     [ ?mirror { t | nil } ]
     [ ?interdigitateChains { t | nil } ]
     [ ?dummyFlexBothEndNets { t | nil } ]
     [ ?chainLeftNet { Source | Drain | Either } ]
     [ ?chainAlignPMOS { Top | Center | Bottom } ]
     [ ?chainAlignNMOS { Top | Center | Bottom } ]
     [ ?allowSingleBulk { t | nil } ]
     [ ?abutStrategy { sdFirst | dummyFirst } ]
     [ ?verbose { t | nil } ]
     )
     => l_abutType / nil
```

Description

Creates an `abutType` DPL, which can be passed to the `gpeCreateAbutEntry` API. The abutment then uses the chaining parameters specified in the `abutType` parameter. The defaults for the arguments are obtained from the Modgen environment if set. Otherwise, the defaults of `lxChain` are honored.

Arguments

`?syncChains { t | nil }`

Synchronizes chains across rows by first arranging the specified instances into rows and then by abutting the instances row-wise from right to left in each column pair.

Note: The `?syncChains` argument uses the row bounding box-Y overlaps to determine the rows that need to be considered for synchronizing the chains.

`?preserveRows { t | nil }`

Accepts multiple rows as input and chains each row using the optional parameters specified.

`?permute { t | nil }`

Permutes instances. The default value is `t`.

`?abutStrategy { sdFirst | dummyFirst }`

Specifies the abutment behavior. Valid values are:

- `sdFirst`: (default) Abuts source or drain first. If this fails, the function tries abutting dummy shapes.
- `dummyFirst`: Abuts dummy shapes first. If this fails, the function tries abutting source or drain.

`?mirrorEquivOrients { t | nil }`

Mirrors to equivalent orientations, for example chaining mirrors `R0` to `MY` and vice versa. The default value is `t`. When set to `nil`, chaining mirrors `MX` to `R180` and vice versa.

`?mirror { t | nil }`

Mirrors instances. The default value is `t`.

`?interdigitateChains { t | nil }`

Identifies and defines the pseudoparallel nets. The default value is `nil`.

`?dummyFlexBothEndNets { t | nil }`

Controls whether the dummy devices can have different source or drain connectivity.

`?chainLeftNet { Source | Drain | Either }`

Specifies the nets that must be optimized to the left of the generated chains. Valid values are:

- `Source`: (default) The generated chain is optimized such that one of its source nets is on the left-hand side of the chain.
- `Drain`: The generated chain is optimized such that one of its drain nets is on the left-hand side of the chain.
- `Either`: Indicates that there are no preferences, and so the default behavior from previous releases is maintained.

The values must be enclosed in quotation marks.

`?chainAlignPMOS { Top | Center | Bottom }`

Controls PMOS chaining alignment. The values must be enclosed in quotation marks. The default is `Top`.

`?chainAlignNMOS { Top | Center | Bottom }`

Controls NMOS chaining alignment. The values must be enclosed in quotation marks. The default is `Bottom`.

`?allowSingleBulk { t | nil }`

Abuts devices if only one of the devices has a bulk and the `?useDeviceOrder` argument is set to `t`. The default value is `t`.

`?verbose { t | nil }` Controls the display of warning messages.

Value Returned

`l_abutType`	Returns the `abutType` DPL created.
`nil`	The command was unsuccessful.

Example

Creates an `abutType` DPL that has synchronized chains and can accept multiple rows as input:

```
abutType = gpeCreateAbutType(?syncChains t ?abutStrategy "dummyFirst")
=> (nil syncChains t abutStrategy "dummyFirst" )
```

gpeCreateAlignment

```
gpeCreateAlignment(
     [ ?direction t_direction ]
     [ ?type t_type ]
     [ ?reflayer g_reflayer ]
     [ ?spacing f_spacing ]
     [ ?verbose { t | nil } ]
     )
     => l_alignment / nil
```

Description

Creates an alignment DPL, which can be passed as an alignment argument for `gpeCreateGridEntry` to set the member alignment and spacing of a grid entry. Specifying a direction, although not necessary for types other than center, is highly recommended.

Arguments

`?direction` *`t_direction`*

Specifies the alignment. Valid values are:

- `horizontal`: A horizontal alignment to be passed as the `hAlign` parameter.
- `vertical`: A vertical alignment to be passed as the `vAlign` parameter.

The direction parameter is always recommended but is only required if the `?type` parameter is set to `center` or `custom`.

`?type` *`t_type`*

Specifies the type of alignment. Valid values are:

- `left`: (default for `horizontal` alignment) Aligns the left sides of instances when the direction is horizontal.
- `right`: Aligns the right sides of instances when the direction is horizontal.
- `center`: Aligns the centers of the instances.
- `top`: (default for `vertical` alignment) Aligns the top sides of instances when the direction is vertical.
- `bottom`: Aligns the bottom sides of instances when the direction is vertical.
- `custom_left`: Creates a spacing from the left side of the instance when the direction is horizontal.
- `custom_right`: Creates a spacing from the right side of the instance when the direction is horizontal.
- `custom`: Equivalent to `custom_bottom` when the direction is vertical and `custom_left` when the direction is horizontal.
- `custom_top`: Creates a spacing from the top of this instance when the direction is vertical.
- `custom_bottom`: Creates a spacing from the bottom of this instance when the direction is vertical.

`?reflayer` *`g_reflayer`*

Specifies the reference layer to which the `gridEntry` must be aligned. You can specify a layer-purpose pair, only a layer name or number, or the layer and purpose numbers. If unspecified, the instance bounding box determined by the Modgen is used.

Example: `list(187 255), list("poly" "drawing")`

Note: Both mature and advanced nodes support layers when applied to a custom spacing-type alignment. In addition, the advanced node release supports use of the `?layer` parameter in combination with the `?type` parameters `left`, `right`, `top`, `bottom`, and `center`.

`?spacing` *`f_spacing`*	Specifies the minimum distance required between the alignment object and the reference instance. This argument can be used only if `?type` is set to one of the `custom*` options.
`?verbose` { `t` | `nil` }	Controls the display of warning messages.

Value Returned

`l_alignment`	Represents the Modgen sandbox alignment DPL that was created.
`nil`	The command was unsuccessful.

Examples

Creates an alignment object in the vertical direction of type `center`:

```
vAlign = gpeCreateAlignment(?direction "vertical" ?type "center")
=> (nil type "center" direction "vertical")
```

Creates an alignment object in the horizontal direction of type `right`:

```
hAlign = gpeCreateAlignment(?direction "horizontal" ?type "right")
=> (nil type "right" direction "horizontal")
```

gpeCreateAlignmentAndSpacing

```
gpeCreateAlignmentAndSpacing(
     u_sandbox
     [?hAlignType { left | right | custom_left | custom_right | center }]
     [?hSpacing f_spacing]
     [?hRefLayer t_layerName]
     [?hRefPurpose t_purposeName]
     [?vAlignType { bottom | top | custom_bottom | custom_top | center }]
     [?vSpacing f_spacing]
     [?vRefLayer t_layerName]
     [?vRefPurpose t_purposeName]
     [?useDefaultForUnspecifiedArgs { t | nil } ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Specifies the alignment and spacing settings for all the grid entries in the given Modgen sandbox object.

Arguments

u_sandbox

Specifies the Modgen sandbox object.

?hAlignType { left | right | custom_left | custom_right | center }

Specifies the horizontal alignment. The default value is `left`.

If not specified, but a horizontal spacing is specified, `custom_left` is considered to be the default.

?hSpacing *f_spacing*

Specifies the horizontal spacing. If unspecified, the default value is `0.0`.

?hRefLayer *t_layerName*

Specifies the name of the layer to be used as the reference to determine the horizontal spacing.

?hRefPurpose *t_purposeName*

Specifies the name of the purpose to be used as the reference to determine the horizontal spacing.

When a horizontal spacing value is specified and the reference layer and purpose are not specified, then the Modgen master bounding box is considered to be the reference.

`?vAlignType` { `bottom` | `top` | `custom_bottom` | `custom_top` | `center` }

Specifies the vertical alignment. The default value is `bottom`.

If the vertical alignment is not specified, but a vertical spacing is specified, `custom_bottom` is considered to be the default.

`?vSpacing` *`f_spacing`*

Specifies the vertical spacing. If unspecified, the default value is `0.0`.

`?vRefLayer` *`t_layerName`*

Specifies the name of the layer to be used as the reference to determine the vertical spacing.

`?vRefPurpose` *`t_purposeName`*

Specifies the name of the purpose to be used as the reference to determine the vertical spacing.

When a vertical spacing value is specified and the reference layer and purpose are not specified, then the Modgen master bounding box is considered to be the reference.

`?useDefaultForUnspecifiedArgs` { `t` | `nil` }

Indicates that if any alignment is not defined, it should be set to its default value. The default value is `nil`. For example, if a new value is not provided for an alignment setting, the existing value is not overwritten.

`?verbose` { `t` | `nil` } Controls the display of warning messages.

Value Returned

`t`	The specified alignment and spacing settings were set.
`nil`	The command was unsuccessful.

Examples

Sets the sbox horizontal alignment to `right` and vertical spacing to `10.0`. The vertical alignment is not specified, and the default `custom_bottom` is used.

```
gpeCreateAlignmentAndSpacing(sbox ?hAlignType "right" ?vSpacing 10.0)
=> t
```

Sets the sbox horizontal alignment to `custom_right` and spacing to `5.0`. The reference is `Metal1 drawing`. The vertical alignment is set to `bottom`.

```
gpeCreateAlignmentAndSpacing(sbox ?hAlignType "custom_right" hSpacing 5.0
?hRefLayer "Metal1" ?hRefPurpose "drawing" ?useDefaultForUnspecifiedArgs t)
=> t
```

Sets the sbox alignment to `left` and `bottom` and spacing to `0.0`, which are the default values.

```
gpeCreateAlignmentAndSpacing(sbox ?useDefaultForUnspecifiedArgs t)
=> t
```

gpeCreateDummyConfig

```
gpeCreateDummyConfig(
     [ ?master t_master]
     [ ?source l_source ]
     [ ?connectivity l_connectivity ]
     [ ?parameters l_parameters ]
     [ ?template l_template ]
     [ ?defaultNet t_defaultNet ]
     [ ?verbose { t | nil } ]
     )
     => l_dummyConfig / nil
```

Description

Creates a `dummyConfig` DPL, which can be used as an argument for `gpeCreateGridEntry` to create a dummy grid entry. If the master is not set, the source is used to determine the master. If the source is not set, the dummy uses the neighboring instances as its source.

Arguments

`?master` *`t_master`*

Specifies the master cellview for creating the `dummyConfig` sandbox DPL. It is in the string format:

`libraryName/cellName/viewName`.

`?source` *`l_source`*

Includes a coordinate list `list(deltaRow deltaCol)` specifying the row offset and column offset from the dummy to the dummy source on the grid. For example, in `list(0 1)`, the instance to the right is considered to be the source.

`?connectivity` *`l_connectivity`*

Defines the connectivity between instance terminals and nets. The format is a list of lists specifying the instance terminals and net names. The argument is typically constructed using repeated calls to the `gpeCreateDummyConfigNet` API. The connectivity parameter takes precedence over `?defaultNet`.

`?parameters` *`l_parameters`*

Specifies the parameters for the `dummyConfig` sandbox object. The format is a list of lists of instance parameter names and their values, typically constructed using repeated calls to the `gpeCreateDummyConfigParam` API.

`?template` *`l_template`*

Specifies the values for the arguments, if not otherwise provided.

`?defaultNet` *`t_defaultNet`*

Specifies the net to which all terminals of the dummy must be connected when their connection is not specified.

`?verbose { t | nil }` Controls the display of warning messages.

Value Returned

`l_dummyConfig`	Returns the `dummyConfig` sandbox DPL that was created.
`nil`	The command was unsuccessful.

Examples

Creates a `dummyConfig` object. The master and source are specified:

```
lcv = "gpdk090/pmos2v/layout"
dummyConfig = gpeCreateDummyConfig(?master lcv ?source '(0 -1))
=> (nil src_index (0 -1) lcv "gpdk090/pmos2v/layout" type "lcv")
```

Creates a `dummyConfig` object by specifying the dummy connectivity. Here, terminal `S` is connected to `7`, terminal `B` is connected to `8`, and rest of the terminals are connected to `10`.

```
net1 = gpeCreateDummyConfigNet("S" ?netName "7")
net2 = gpeCreateDummyConfigNet("B" ?netName "8")
connectivity = list(net1 net2)
dc = gpeCreateDummyConfig(?defaultNet "10" ?connectivity connectivity)
de = gpeCreateGridEntry(?row 0 ?col 1 ?orientation "R0" ?dummyConfig dc)
```

Creates a `dummyConfig` object with terminal `S` connected to `7`, terminal `B` connected to `8`, and rest of the terminals are connected to `9` (because `allTerms` is specified).

```
net1 = gpeCreateDummyConfigNet("allTerms" ?netName "9")
net2 = gpeCreateDummyConfigNet("S" ?netName "7")
net3 = gpeCreateDummyConfigNet("B" ?netName "8")
connectivity = list(net1 net2 net3)
dc = gpeCreateDummyConfig(?defaultNet "10" ?connectivity connectivity)
de = gpeCreateGridEntry(?row 0 ?col 1 ?orientation "R0" ?dummyConfig dc)
```

gpeCreateDummyConfigNet

```
gpeCreateDummyConfigNet(
     t_termName
     [ ?netName t_netName ]
     [ ?verbose { t | nil } ]
     )
     => l_dummyConfigNet / nil
```

Description

Creates a `dummyConfigNet` DPL that can be added to a list with other `dummyConfigNet` DPLs and passed as the `connectivity` argument to `gpeCreateDummyConfig` to create a `dummyConfig` DPL.

Arguments

t_termName	
	Specifies the name of the instance terminal for which the dummy net is being created. Use `allTerms` to indicate all `instTerms`.
?netName *t_netName*	
	Specifies the name of the net to which the above `instTerm` must be connected.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

l_dummyConfigNet	Returns the `dummyConfigNet` DPL created.
`nil`	Could not create the `dummyConfigNet` DPL.

Example

Creates a `dummyConfigNet` DPL based on the specified `InstTerm` and `netName`:

```
dummyConfigNet = gpeCreateDummyConfigNet("G" ?netName "GND")
=> (nil type "specify" value "GND" name "G")
```

gpeCreateDummyConfigParam

```
gpeCreateDummyConfigParam(
     t_name
     [ ?type t_type ]
     [ ?value t_value ]
     [ ?verbose { t | nil } ]
     )
     => l_dummyConfigParam / nil
```

Description

Creates a `dummyConfigParam` DPL that can be added to a list with other `dummyConfigParam` DPLs and passed as the `parameters` argument to `gpeCreateDummyConfig` to create a `dummyConfig` DPL.

Arguments

t_name	Specifies the name of a `cdfParameter` or the name of the parameter placeholder that is mapped by `ciMapParam`.
`?type` *t_type*	Controls how the parameter can be set. Valid values are: ■ `default`: Uses the default `cdfParameter` value, which is the minimum. ■ `specify`: Lets you specify a value. ■ `match_src`: Copies the parameter from the dummy source.
`?value` *t_value*	Specifies the value to be applied to the `cdfParameter` when `?type` is set to `specify`.
`?verbose` { `t` | `nil` }	Controls the display of warning messages.

Value Returned

l_dummyConfigParam	Returns the dummyConfigParam DPL created. Example: (nil name "l" type "specify" value "45n")
nil	Could not create the dummyConfigParam DPL.

Example

Creates a dummyConfigParam DPL with the specified name. The ?type is set to specify and the value is set to 45n:

```
dummyConfigParam = gpeCreateDummyConfigParam("I" ?type "specify" ?value "45n")
=> (nil value "45n" type "specify" name "I")
```

In the following example, ?type is set to default:

```
dummyConfigParam = gpeCreateDummyConfigParam("I" ?type "default")
=> (nil value "" type "default" name "I")
```

gpeCreateGridEntry

```
gpeCreateGridEntry(
     [ ?row x_row ]
     [ ?col x_col ]
     [ ?name t_name ]
     [ ?dummyConfig l_dummyConfig ]
     [ ?orientation t_orientation ]
     [ ?hAlign l_hAlign ]
     [ ?vAlign l_vAlign ]
     [ ?template l_template ]
     [ ?verbose { t | nil } ]
     )
     => l_gridEntry / nil
```

Description

Creates a Modgen sandbox grid entry DPL for an instance or a dummy. A grid entry can represent either an instance or a dummy.

Arguments

`?row` *x_row*

Specifies the grid entry row index.

`?col` *x_col*

Specifies the grid entry column index.

`?name` *t_name*

Specifies the name of the symbol or schematic that contains the instance.

`?dummyConfig` *l_dummyConfig*

Specifies the dummy configuration returned by running the `gpeCreateDummyConfig` function.

Note: Specify either `?name` or `?dummyConfig` because these arguments are mutually exclusive.

`?orientation` *t_orientation*

Specifies the orientation of the entry. Valid values are: `"R0"`, `"R90"`, `"R180"`, `"R270"`, `"MX"`, `"MY"`, `"MXR90"`, `"MYR90"`.

`?hAlign` *`l_hAlign`*

Indicates a horizontal alignment of the instance returned by running `gpeCreateAlignment`.

`?vAlign` *`l_vAlign`*

Indicates a vertical alignment of the instance returned by running `gpeCreateAlignment`.

`?template` *`l_template`*

Specifies the value for arguments, if not provided.

`?verbose` { `t` | `nil` } Controls the display of warning messages.

Value Returned

`l_gridEntry`	Returns the Modgen sandbox grid entry DPL created.
`nil`	Could not create a Modgen sandbox grid entry DPL.

Example

In the following examples, Modgen sandbox grid entry DPLs are created based on the specified arguments. Here, a grid entry DPL is created for an instance with name M9:

```
gridEntry = gpeCreateGridEntry(?row 0 ?col 0 ?name "M9" ?orientation "R270")
=> (nil name "M9" col 0 row 0 orient "R270")
```

Here, a grid entry DPL is created for a dummy, which is configured according to the `dummyConfig` DPL:

```
dummyConfig = gpeCreateDummyConfig(?source '(0 1))
=> (nil src_index (0 1) type "lcv")
dummyEntry = gpeCreateGridEntry(?row 0 ?col 0 ?dummyConfig dummyConfig)
=> (nil col 0 row 0 symbol "*" type "dummy" dummyConfig (nil src_index (0 1) type
"lcv"))
```

gpeCreateMapEntry

```
gpeCreateMapEntry(
     [ ?schematicName t_schematicName ]
     [ ?symbolName t_symbolName ]
     [ ?template l_template ]
     [ ?verbose { t | nil } ]
     )
     => l_mapEntry / nil
```

Description

Creates a map entry DPL that stores the schematic name of an instance, along with its symbol mapping.

Arguments

`?schematicName` *t_schematicName*

Specifies the name of the schematic instance.

`?symbolName` *t_symbolName*

Specifies the symbol mapping of the instance.

`?template` *l_template*

Specifies a map entry, which serves as a template. Provides the values for schematic and symbol names if not otherwise specified.

`?verbose { t | nil }` Controls the display of warning messages.

Value Returned

l_mapEntry	Returns the map entry DPL that was created.
`nil`	A Modgen sandbox map entry could not be created.

Example

Creates a map entry DPLs:

```
mapEntry1 = gpeCreateMapEntry(?schematicName "M9" ?symbolName "B")
=> (nil symbol "B" schName "M9")
```

```
templateEntry = gpeCreateMapEntry(?schematicName "M1")
=> (nil schName "M1")
```

Creates a map entry DPL with a template:

```
mapEntry2 = gpeCreateMapEntry(?symbolName "C" ?template templateEntry)
=> (nil symbol "C" schName "M1")
```

gpeCreateSandbox

```
gpeCreateSandbox(
     [ ?cv d_cellview ]
     [ ?ids l_figs ]
     [ ?sync { t | nil } ]
     [ ?verbose { t | nil } ]
     )
     => u_sandbox / nil
```

Description

Creates a Modgen sandbox object that includes the specified instances. This function can be used to create Modgen sandbox objects in both schematic and layout cellviews. If none of the arguments are provided, the current edit cellview and selected instances are considered.

Changes made to a Modgen sandbox object are updated to the associated Modgen constraint and layout placement only when the Modgen sandbox object is in the synchronization mode. Either set the `?sync` argument to `t` or use the `gpeSyncSandbox` API to turn on the synchronization mode. After customizing the Modgen sandbox object as per your requirements, use the `gpeFinishSandbox` SKILL function to delete the Modgen sandbox object and its associated scratch cellview and transfer the Modgen constraint to the schematic view.

Arguments

`?cv` *`d_cellview`*

Specifies the cellview ID. If unspecified, uses the current cellview.

`?ids` *`l_figs`*

Specifies a list of instances to be included in the Modgen sandbox.

`?sync { t | nil}`

Turns on synchronization mode for the Modgen sandbox object created. The default is `nil`.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

u_sandbox	Returns the Modgen sandbox object created.
nil	The Modgen sandbox object could not be created.

Examples

In the following example, a Modgen sandbox is created using the current cellview:

```
sbox = gpeCreateSandbox()
```

In the following example, cv is a list of cellviews returned by geGetEditCellView and instList are the instances returned by geGetSelectedSet. These values are passed as arguments to create a Modgen sandbox object:

```
cv = geGetEditCellView()
instList = geGetSelectedSet()
sb = gpeCreateSandbox(cv instList)
The cv references
```

Creates a Modgen that includes the specified instances.

```
sbox = gpeCreateSandbox( ?sync t);
gpeSetMap(sbox list( list("M0" "X") list("M1" "Y")));
gpeSetGrid(sbox list( "X" "X") list("Y" "Y")));
```

gpeDeleteDummyEntries

```
gpeDeleteDummyEntries(
     u_sandbox
     g_includeInternal
     g_unplaceOnly
     g_fullRCOnly
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Deletes dummy entries from the given Modgen sandbox object and compresses the grid based on the specified options.

Arguments

u_sandbox	Specifies the Modgen sandbox object.

g_includeInternal

Specifies whether internal dummies must be deleted. Valid values are `t` and `nil`.

When set to `t`, internal dummies are deleted. Internal dummies are the dummies present between instances.

Example of internal dummies:

```
*   *   A B

A   C   D B
```

When set to `nil`, only the non-internal dummies are deleted. Non-internal dummies are the dummies that form full rows or columns at the edges of a Modgen.

Examples of non-internal dummies:

Row of non-internal dummies:

```
*   *   *   *

A   A   B   B

C   D   A   A
```

Row and column of non-internal dummies:

```
*   *   *   *   *

A   A   B   B   *

C   D   A   A   *

*   *   *   *   *
```

g_unplaceOnly

Unplaces the available dummies. Does not remove the rows or columns. Valid values are `t` and `nil`.

g_fullRCOnly

Deletes rows or columns that are either empty or contain dummies. Valid values are `t` and `nil`.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`t`	The dummy entries were deleted.
`nil`	The command was unsuccessful.

Example

Removes all rows and columns that are filled with either dummies or empty cells surrounding a core of instances. The grid is then compressed.

```
sbox = gpeEditSandbox()
gpeDeleteDummyEntries(sbox nil nil t)
> t
```

gpeDeleteSandbox

```
gpeDeleteSandbox(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Deletes the specified Modgen sandbox object but does not delete its associated figGroup.

Arguments

u_sandbox	Specifies the ID of the Modgen sandbox object to be deleted.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The Modgen sandbox object was deleted.
nil	The Modgen sandbox object could not be deleted.

Example

In the following example, `cv` represents the cellview returned by `geGetCurrentCellView` and `inst1` and `inst2` are the instances returned by `dbFindAnyInstByName`. These values are passed as arguments to create a Modgen sandbox object `sbox`. `gpeDeleteSandbox` is then used to delete this Modgen sandbox object.

```
cv=geGetCurrentCellView()
inst1=dbFindAnyInstByName(cv "M1")
inst2=dbFindAnyInstByName(cv "M2")
sbox=gpeCreateSandbox(?cv  cv ?ids list(inst1 inst2))

gpeDeleteSandbox(sbox)
```

gpeEditSandbox

```
gpeEditSandbox(
     [ d_modgenID | d_figGroupID ]
     [ g_sync ]
     [ ?verbose { t | nil } ]
     )
     => u_sandbox / nil
```

Description

Returns a Modgen sandbox object that represents the current Modgen or figGroup. The sandbox object can be edited using SKILL APIs.

Changes made to a Modgen sandbox object are updated to the associated Modgen constraint and layout placement only when the Modgen sandbox object is in the synchronization mode. Either set the `?sync` argument to `t` or use the `gpeSyncSandbox` API to turn on the synchronization mode. After customizing the Modgen sandbox object as per your requirements, use the `gpeFinishSandbox` SKILL function to delete the Modgen sandbox object and its associated scratch cellview and transfer the Modgen constraint to the schematic view.

Arguments

d_modgenID | *d_figGroupID*

Specifies the figGroup ID or Modgen constraint ID. The default is the selected Modgen figGroup then the selected Modgen constraint.

g_sync

Specifies whether the sandbox is in the synchronized state. The default value is `nil`.

?verbose { t | nil }

Controls the display of warning messages.

Value Returned

u_sandbox	Returns the Modgen sandbox object.
nil	The Modgen sandbox object could not be found.

Example

In the following example, the editing mode is turned on for the Modgen sandbox that is present in the current cellview:

```
sb = gpeEditSandbox()
```

In the following example, the values for `fg` and `sync` are first instantiated. These values are then used in `gpeEditSandbox` to turn on editing mode for corresponding Modgen sandbox objects:

```
fg = car(exists(x geGetSelectedSet() (x~>objType=="figGroup" &&
x~>type=="modgen")))
sync = nil
sb = gpeEditSandbox(fg sync)
```

gpeFinishSandbox

```
gpeFinishSandbox(
     u_sandbox
     [ ?transfer { t | nil } ]
     [ ?removeScratch { t | nil } ]
     [ ?backAnnotate { t | nil } ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Deletes the specified Modgen sandbox object. If run from the schematic view, the Modgen's associated scratch cellview is also deleted. If run from the layout view, the command optionally transfers the Modgen constraint to the schematic view and backannotates any dummies. `gpeFinishSandbox` must be used in tandem with the functions that create sandboxes, such as `gpeCreateSandbox` and `gpeEditSandbox`. It is recommend that all Modgen sandbox objects are finished by calling this function.

Arguments

u_sandbox	Specifies the Modgen sandbox object to be deleted.
`?transfer { t | nil }`	Specifies whether the Modgen constraint must be transferred to its schematic view. The default value is `nil`.
`?removeScratch { t | nil}`	Specifies whether the scratch cellview must be deleted. The default value is `t`.
`?backAnnotate { t | nil }`	Specifies whether all dummies must be backannotated. This value is valid only if `g_transfer` is set to `t`. The default value is `nil`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	The specified Modgen sandbox object was deleted.
`nil`	The command was unsuccessful.

Example

In the following example, `cv` is the cellview returned by `geGetCurrentCellView`, and inst1 and inst2 are the instances returned by `dbFindAnyInstByName`. These values are passed as arguments to create a Modgen sandbox object `sbox`. `gpeFinishSandbox` is then used to delete this Modgen sandbox object. The Modgen constraint is transferred to the schematic view, the scratch cellview is deleted, and all dummies are backannotated.

```
cv = geGetCurrentCellView()
inst1 = dbFindAnyInstByName(cv "M1")
inst2 = dbFindAnyInstByName(cv "M2")
sbox = gpeCreateSandbox(?cv  cv ?ids list(inst1 inst2))
gpeFinishSandbox(sbox ?transfer t ?removeScratch t ?backAnnotate t)
```

gpeGetAbut

```
gpeGetAbut(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => l_abut / nil
```

Description

Returns the Modgen abutment DPL for the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_abut	Returns a Modgen sandbox abutment DPL, which is a list of chains.
nil	The command was unsuccessful.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. This value is passed as an argument to `gpeGetAbut`:

```
sb = gpeEditSandbox()
abutment = gpeGetAbut(sb)
```

gpeGetAbutTypeList

```
gpeGetAbutTypeList(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => l_abutTypes / nil
```

Description

Returns a list of abutment types for the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_abutTypes	Returns a list of abutment types in the format: `'( '(abutTypeName1 abutType1) '(abutTypeName2 abutType2) ... )`
nil	The command was unsuccessful.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. This value is passed as an argument to `gpeGetAbutTypeList`:

```
sb = gpeEditSandbox()
gpeGetAbutTypeList(sb ?verbose t)
> (("type0" 0) ("type1" 1))
```

gpeGetAbutTypeName

```
gpeGetAbutTypeName(
     u_sandbox
     x_abutType
     [ ?verbose { t | nil } ]
     )
     => t_abutTypeName / nil
```

Description

Returns the name corresponding to the given abutment type number from the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
x_abutType	Specifies the abutment type number.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t_abutTypeName	Returns the abutment type name corresponding to the specified abutment type number.
nil	The command was unsuccessful.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. This value is passed as an argument to `gpeGetAbutTypeName`:

```
sb = gpeEditSandbox()
gpeGetAbutTypeName(sb 0 ?verbose t)
>"type0"
```

gpeGetConstraints

```
gpeGetConstraints(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => lu_constraintIDs / nil
```

Description

Returns a list of constraint IDs that are associated with the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

lu_constraintIDs	Returns a list of constraint IDs.
nil	Could not find any matching constraints.

Example

In the following example, the value for `sb` is determined by calling `gpeEditSandbox`. This value is then passed as an argument for `gpeGetConstraints`:

```
sb = gpeEditSandbox()
ciCon = car(gpeGetConstraints(sb))
```

gpeGetDefaultAbutType

```
gpeGetDefaultAbutType(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t_abutTypeName / nil
```

Description

Returns the default abutment type name for the given Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t_abutTypeName	Returns the default abutment type name.
nil	The command was unsuccessful.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. This value is passed as an argument to `gpeGetDefaultAbutType`:

```
sb = gpeEditSandbox()
gpeGetDefaultAbutType(sb ?verbose t)
> "type0"
```

gpeGetFigGroup

```
gpeGetFigGroup(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => d_figGoupID / nil
```

Description

Returns the ID of the Modgen figGroup that is associated with the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

d_figGroupID	Returns the figGroup ID of the Modgen associated with the specified Modgen sandbox object.
nil	Could not find a matching Modgen figGroup.

Example

In the following example, the value for `sb` is determined by calling `gpeEditSandbox`. This value is then passed as argument for `gpeGetFigGroup`:

```
sb = gpeEditSandbox()
fg = gpeGetFigGroup(sb)
```

gpeGetGrid

```
gpeGetGrid(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => l_grid / nil
```

Description

Returns the Modgen sandbox grid DPL that describes the grid placement of the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_grid	Returns the grid DPL.
nil	The command was unsuccessful.

Example

In the following example, the value for `sb` is determined by calling `gpeEditSandbox`. This value is then passed as argument for `gpeGetGrid`:

```
sb   = gpeEditSandbox()
grid = gpeGetGrid(sb)
```

gpeGetGridColCount

```
gpeGetGridColCount(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => x_columnCount / nil
```

Description

Returns the number of columns available in the grid of the given Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

x_columnCount	Number of columns in the grid of the given Modgen sandbox object.
nil	The command was unsuccessful.

Example

In the following example, a Modgen sandbox object is instantiated. `gpeGetGridColCount` is then used to retrieve the number of columns:

```
sbox = gpeStartSandbox()  => pe:0x####
gpeGetGridColCount(sbox)  => 3
```

gpeGetGridEntry

```
gpeGetGridEntry(
     u_sandbox
     [ ?ref d_refID ]
     [ ?index l_index ]
     [ ?name t_layoutName ]
     [ ?verbose { t | nil } ]
     )
     => l_gridEntry / nil
```

Description

Returns a grid entry that matches the specified argument for the specified sandbox object. When calling this function, only one search argument of type reference, index, or layout name must be specified. If multiple arguments are specified, an error message is displayed.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?ref *d_refID*	Specifies the reference database instance ID of the grid entry.
?index *l_index*	Specifies the index list containing a grid row and column, for example `list(0 0)`.
?name *t_layoutName*	Specifies the layout name of the grid entry.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_gridEntry	Returns the Modgen grid entry (DPL) that matches the specified arguments.
nil	The command was unsuccessful.

Examples

Gets the grid entry that has a matching reference ID.

```
sb = gpeEditSandbox()
window = hiGetCurrentWindow()
cv = deOpenCellView(window)
inst = dbFindAnyInstByName(cv "|M9")
gpeGetGridEntry(sb ?ref inst)
```

Gets the grid entry that has a matching grid index.

```
gpeGetGridEntry(sb ?index list(1 1))
```

Gets the grid entry that has a matching layout name.

```
gpeGetGridEntry(sb ?name "|M9")
```

gpeGetGridEntries

```
gpeGetGridEntries(
     u_sandbox
     [ ?row x_row ]
     [ ?col x_col ]
     [ ?name t_name ]
     [ ?verbose { t | nil } ]
     )
     => l_gridEntries / nil
```

Description

Returns a list of grid entries from the Modgen sandbox that match the specified identifiers. If you specify multiple identifiers, then the function returns a list of instances that satisfy all requirements.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?row *x_row*	Identifies the row from which all *l_gridEntries* must be retrieved.
?col *x_col*	Identifies the column from which all *l_gridEntries* must be retrieved. You can use this argument along with ?row to get a single entry.
?name *t_name*	Specifies the symbol, schematic, or layout name from which *l_gridEntries* must be retrieved.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_gridEntries	Returns one or more Modgen sandbox grid entries in the form of a DPL with parameters: col, row, type, orient, symbol, name, inst, dummyConfig, hAlign, vAlign.
nil	The command was unsuccessful.

Example

In the following example, the value of sb is determined by calling gpeEditSandbox. This value is then passed as an argument in three different examples. In the first example, the Modgen sandbox object and the row and column values are specified.

```
sb = gpeEditSandbox()
gpeGetGridEntries(sb ?row 0 ?col 1)
```

In the following example, the argument list includes the Modgen sandbox object and the schematic name.

```
gpeGetGridEntries(sb ?name "PM1.18")
```

Here, the argument list includes the Modgen sandbox object and the row number.

```
gpeGetGridEntries(sb ?row 1)
```

In the following example, the argument list includes the Modgen sandbox object and the column number.

```
gpeGetGridEntries(sb ?col 1)
```

gpeGetGridRowCount

```
gpeGetGridRowCount(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => x_rowCount / nil
```

Description

Returns the number of rows available in the grid of the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

x_rowCount	Number of rows in the grid of the specified Modgen sandbox object.
nil	The command was unsuccessful.

Example

In the following example, a Modgen sandbox object is instantiated. `gpeGetGridRowCount` is then used to retrieve the number of rows:

```
sbox = gpeStartSandbox()  => pe:0x####
gpeGetGridRowCount(sbox)  => 3
```

gpeGetGridSelection

```
gpeGetGridSelection(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => l_gridSelection / nil
```

Description

Returns a list of indexes that represent the grid selection in the GPE (Grid Pattern Editor) assistant. The DPL is a list of list of indexes, for example `((0 1) (1 3) ... )`, that are associated with the specified `userType` (UT).

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_gridSelection	Returns a list of indexes that represent the grid selection in the associated GPE.
nil	The command was unsuccessful.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. This value is then passed as argument for `gpeGetGridSelection`. The return value is a list of grid selections that include indexes from `(1 4)` to `(1 7)`, indexes from `(0 5)` to `(0 7)`, grid cell in `(0 2)`, and grid cell in `(1 1)`.

```
sb = gpeEditSandbox(s)
selection = gpeGetGridSelection(sb)
( ((1 4) (1 7)) ((0 5) (0 7)) (0 2) (1 1) )
```

gpeGetMap

```
gpeGetMap(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => l_map / nil
```

Description

Returns the Modgen sandbox map DPL for the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_map	Returns the Modgen sandbox map DPL, which is a list of mapEntries.
nil	The Modgen sandbox map DPL could not be retrieved.

Example

In the following example, the value for sb is determined by calling gpeEditSandbox. This value is then passed as argument for gpeGetMap:

```
sb  = gpeEditSandbox()
map = gpeGetMap(sb)
```

gpeGetMapEntry

```
gpeGetMapEntry(
     u_sandbox
     t_name
     [ ?verbose { t | nil } ]
     )
     => l_mapEntry / nil
```

Description

Returns the map entry DPL from the specified Modgen sandbox object that matches the specified schematic or symbol name. The function first looks for a matching schematic name; if not found, then looks for a matching symbol name and returns the first match.

Arguments

u_sandbox	Specifies the Modgen sandbox object from which map entries must be retrieved.
t_name	Specifies the schematic or symbol name for which the map entry must be retrieved.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_mapEntry	Returns a disembodied property list with parameters: `type`, `schName`, `symbol`, `availableInsts`, `placedCount`, `totalCount`.
`nil`	A map entry could not be found.

Example

Returns the map entry DPL from the specified Modgen sandbox object.

```
sb = gpeEditSandbox() => pe:0x#####
me = gpeGetMapEntry(sb "M0")
```

gpeInsertEmptyColumns

```
gpeInsertEmptyColumns(
     u_sandbox
     t_direction
     x_startColumn
     [ ?count n_count ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Inserts empty columns in the specified Modgen sandbox object in the specified direction, starting at the specified column.

Arguments

u_sandbox	Specifies the Modgen sandbox object in which empty columns are to be inserted.
t_direction	Specifies the direction in which the additional columns are to be inserted relative to the start column. Valid options are `left` and `right`.
x_startColumn	Specifies the reference column for inserting new columns. The column indexes start at 0.
`?count` *n_count*	Specifies the number of columns to be inserted. The default value is 1.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Empty columns were inserted as specified.
`nil`	The command was unsuccessful.

Examples

Inserts 5 empty columns after the last column.

```
sbox = [create and populate a sandbox]
gpeInsertEmptyColumns(sbox "right" difference(gpeGetGridColCount(sbox) 1) ?count
5)
```

Inserts 5 empty columns to the left.

```
gpeInsertEmptyColumns(sbox "left" 0 ?count 5)
```

Inserts 1 empty column to the left of the 4th column, between columns indexed as 2 and 3.

```
;; (index 3 is the 4th column)
gpeInsertEmptyColumns(sbox "left" 3)
```

Inserts one column to the right of the third column, between columns indexed as 2 and 3.

```
;; (index 2 is the 3rd column)
gpeInsertEmptyColumns(sbox "right" 2)
```

gpeInsertEmptyRows

```
gpeInsertEmptyRows(
     u_sandbox
     t_direction
     x_startRow
     [ ?count n_count ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Inserts empty rows in the specified Modgen sandbox object in the specified direction, starting at the specified row.

Arguments

u_sandbox	Specifies the Modgen sandbox object in which empty rows must be inserted.
t_direction	Specifies the direction in which the additional rows must be inserted relative to the start row. Valid options are `top` and `bottom`.
x_startRow	Specifies the reference row for inserting new rows. New rows are inserted above or below the reference row. The row indexes start at 0.
`?count` *n_count*	Specifies the number of rows to be inserted. The default value is 1.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Empty rows were inserted as specified.
`nil`	The command was unsuccessful.

Examples

Inserts 5 empty rows below the last row.

```
sbox = [create and populate a sandbox]
gpeInsertEmptyRows(sbox "top" difference(gpeGetGridRowCount(sbox) 1) ?count 5)
```

Inserts 5 empty rows above the first row.

```
gpeInsertEmptyRows(sbox "bottom" 0 ?count 5)
```

Inserts 1 empty row above the fourth row, between the rows indexed as 2 and 3.

```
;; (index 3 is the 4th row)
gpeInsertEmptyRows(sbox "bottom" 3)
```

Inserts 1 empty row below the third row, between the rows indexed as 2 and 3.

```
;; (index 2 is the 3rd row)
gpeInsertEmptyRows(sbox "top" 2)
```

Inserts 1 empty row above the fourth row.

```
gpeInsertEmptyRows(sbox "bottom" 3)
```

Inserts 1 empty row above the fourth row, but suppresses the error messages.

```
gpeInsertEmptyRows(sbox "bottom" 3 ?verbose nil)
```

gpeIsSandboxSync

```
gpeIsSandboxSync(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Checks whether the specified Modgen sandbox object is in sync with its associated Modgen constraint, and returns a Boolean that indicates the status.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The Modgen sandbox object and constraint are synchronized.
nil	The Modgen sandbox object and constraint are not synchronized.

Example

In the following example, the value for `sb` is determined by calling `gpeEditSandbox`. This value is then passed as argument for `gpeIsSandboxSync`:

```
sb = gpeEditSandbox() => pe:0x#####
gpeIsSandboxSync(sb) => nil
```

gpeIsValidAbutType

```
gpeIsValidAbutType(
     u_sandbox
     { t_abutTypeName | x_abutType }
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Validates the specified abutment type name or number for the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
t_abutTypeName | *x_abutType*	Specifies the abutment type name or abutment type number.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The specified abutment type name or number is valid.
nil	The specified abutment type name or number is invalid.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. This value is passed as an argument to `gpeIsValidAbutType`, which is used to validate different abutment type names and numbers:

```
sb = gpeEditSandbox()
gpeIsValidAbutType(sb "type0" ?verbose t)
> t
gpeIsValidAbutType(sb 0 ?verbose t)
> t
gpeIsValidAbutType(sb 2)
```

```
> nil
t_abutTypeName
```

gpePlaceGridEntries

```
gpePlaceGridEntries(
     u_sandbox
     l_gridEntries
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Places the specified grid entries at the specified location in the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
l_gridEntries	Specifies the list of grid entries to be added at the specified location in the sandbox grid. Example: `list(g_gridEntry1 g_gridEntry2 ... )`
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	The specified grid entries were added.
`nil`	The specified grid entries could not be added.

Example

In the following example, the value of `sb` is determined by calling `gpeEditSandbox`. The synchronization mode is turned on for `sb`. `gpeClearGridEntries` is used to unplace a few grid entries in `sb`. `gpePlaceGridEntries` is then used to place these grid entries.

```
> sb = gpeEditSandbox()
> entries = gpeClearGridEntries( sb ?row 1 ?verbose t)
> gpePlaceGridEntries(sb entries ?verbose t)
> t
> gpeSyncSandbox(sb)
```

gpeRemoveEmptyRowsAndColumns

```
gpeRemoveEmptyRowsAndColumns(
     u_sandbox
     g_includeInternal
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Removes empty rows and columns from the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
g_includeInternal	
	When set to `t`, removes all empty rows and columns from the Modgen sandbox object.
	When set to `nil`, removes all empty rows until the first non-empty location is found in the Modgen sandbox object.
`?verbose { t | nil }`	
	Controls the display of warning messages.

Value Returned

`t`	Empty rows and columns were removed from the specified Modgen sandbox object.
`nil`	The command was unsuccessful.

Example

Removes all empty rows and columns from the Modgen sandbox object `sbox`:

```
sbox = gpeEditSandbox()
gpeRemoveEmptyRowsAndColumns(sbox t)
> t
```

gpeSboxp

```
gpeSboxp(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Checks whether the specified object is a valid Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The specified object is a valid Modgen sandbox object.
nil	The specified object is not a valid Modgen sandbox object.

Example

Creates a Modgen sandbox object and assigns it to `sb`. The `gpeSboxp` function is then used to determine if this Modgen sandbox object is valid. Return value `t` indicates that `sb` is a valid Modgen sandbox object:

```
sb = gpeCreateSandbox()
=> pe:0x#####
gpeSboxp(sb)
=> t
```

The value of `sb` is set to `nil`. The `gpeSboxp` function is then used to determine if this Modgen sandbox object is valid. Return value `nil` indicates that `sb` is not a valid Modgen sandbox object:

```
sb = nil
gpeSboxp(sb)
=> nil
```

gpeSetAbut

```
gpeSetAbut(
     u_sandbox
     g_abut
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Sets the abutment for the entire grid, without using individual abutment entries. The entire abutment object of the sandbox-in-place is replaced.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
g_abut	Specifies the abutment status. Valid values are: ■ A Modgen sandbox abutment DPL. ■ `t` to abut all instances. ■ `nil` to unabut all instances.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Indicates that the abutment settings were set as specified.
`nil`	The command was unsuccessful.

Example

In the following example, the value for `sbox` is assigned to the value returned by `gpeEditSandbox`; the value of `abut` is assigned to value returned by `gpeGetAbut`. These values are then passed as arguments to `gpeSetAbut`:

```
sbox = gpeEditSandbox()
abut = gpeGetAbut(sbox)
gpeSetAbut(sbox abut)
```

```
prevAbut = gpeGetAbut(sbox)
```

In the following example, *g_abut* is set to `t`, and so all instances are abutted:

```
gpeSetAbut(sbox t)
```

In the following example, *g_abut* is set to `nil`, and so all instances are unabutted:

```
gpeSetAbut(sbox nil)
```

The following example uses `prevAbut`, a Modgen sandbox DPL, which contains the current abutment information from the sandbox:

```
gpeSetAbut(sbox prevAbut)
```

gpeSetAbutEntries

```
gpeSetAbutEntries
     u_sandbox
     l_abutEntries
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Modifies the Modgen sandbox object by applying the abutment specified by the abutment entries. If there is any conflict between the new entry and any existing entry, the existing entry is removed and the new entry is used.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
l_abutEntries	Represents a list of abut entries that specify the rows and columns to be abutted.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The Modgen sandbox object was modified.
nil	The command was unsuccessful.

Example

The following example first defines the two arguments sbox and `abutEntries`, and then uses them in `gpeSetAbutEntries` to modify the sandbox object.

```
sbox = gpeEditSandbox()
abutEntries = gpeCreateAbutEntries(?rows 0 ?cols list(list(0 2)) ?abutType "type0")
gpeSetAbutEntries(sbox abutEntries)
=> t
```

gpeSetGrid

```
gpeSetGrid(
     u_sandbox
     l_grid
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Sets the entire grid without using individual grid entries. You can also use this function to update grid placement by providing a list of lists describing the row placement of instances.

Note: Specifying value `nil` for *`g_grid`* clears all grid entries.

Arguments

u_sandbox Specifies a Modgen sandbox object.

l_grid Specifies the new DPL or a list of lists that contains the entire pattern. The first list is a list of rows. Each row list is a list of names of symbols, column-by-column. A sandbox grid is created based on this one argument.

Example:

```
grid = list(
list("A" "A" "A" "A" "A")
list("A" "A" "B" "A" "A")
list("A" "B" "C" "B" "A")
list("A" "A" "B" "A" "A")
list("A" "A" "A" "A" "A")
)
```

Rows are specified from row 0 upward, where row 0 is the first row, followed by row 1, and so on. The following special characters are valid:

- `"*"`: Indicates the dummy symbol. Example: `list(list("A" "*" "A" "A") list("B" "B" "*" "B"))`
- `""`, `" "`, `"-"`, `"\t"`, and `"nil"`: Indicate an empty cell in the grid. Example: `list(list("A" "*" "A" nil "" "-" "\t" "A") list("-" nil "\n" "" " " "B" "B" "B"))`
- `"nil"` as a row value: Indicates that the row is empty. Example: `list(list("A" "*" "A" "A") nil list("B" "B" "*" "B"))`

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

t	The Modgen sandbox grid was created.
nil	Could not create a Modgen sandbox grid.

Example

The value for sbox is assigned to the value returned by gpeEditSandbox. grid1 identifies a sandbox grid DPL which holds the current grid placement of sbox. grid2 is a grid created using the specified arguments:

```
sbox = gpeEditSandbox()
=> pe:#####
grid1 = gpeGetGrid(sbox)
grid2 = list( list("A" "A" "B") list("D" "B" "C") list("B" "C" "D") )
```

In the following example, a Modgen sandbox object is created based on values of sbox and grid2 defined above:

```
gpeSetGrid(sbox grid2)
=> t
```

In the following example, a Modgen sandbox object is created based on values of sbox and grid1 defined above:

```
gpeSetGrid(sbox grid1)
=> t
```

gpeSetGridEntry

```
gpeSetGridEntry(
     u_sandbox
     l_gridEntry
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Adds the specified grid entry to the `grid` portion of the specified Modgen sandbox object. If the grid location specified in the grid entry is already occupied, the existing entry is unplaced, and the specified grid entry is set.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
l_gridEntry	Specifies the Modgen sandbox grid entry DPL to be added. The grid entry must have a row, column, and at least one instance specifier such as a symbol, schematic name, or `dummyConfig`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	The grid DPL was updated.
`nil`	The grid DPL could not be updated.

Example

In the following example, a grid entry is created using `gpeCreateGridEntry`. The return value is assigned to `gridEntry`. This value and an `sbox` value are passed as arguments to `gpeSetGridEntry`:

```
gridEntry = gpeCreateGridEntry(?row 1 ?col 1 ?name "B")
=> (nil name "B" col 1 row 1)
gpeSetGridEntry(sbox gridEntry)
=> t
```

gpeSetMap

```
gpeSetMap(
     u_sandbox
     g_map
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Updates the symbol mapping without using individual map entries. When a sandbox map DPL is specified, the current mapping of instances is replaced with the mapping specified by the provided map DPL. Symbol mapping can also be updated by providing a list of lists of strings. The first string in each sub-list holds the schematic name of an instance and the second string holds its updated mapping.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
g_map	Specifies the new mapping information. The value can be either a Modgen sandbox map DPL or a list of lists of strings, where the first string is the schematic name and the second string is the layout name. Example: `list(list("NM0" "A") list("NM1" "B") list("NM2" "C"))`
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Indicates that the Modgen sandbox map DPL was set or updated.
`nil`	Indicates that the Modgen sandbox map DPL could not be set or updated.

Example

Here, `sbox` is assigned to the value returned by `gpeEditSandbox`. `map1` and `map2` identify Modgen sandbox map DPLs. The value for `map1` is determined by calling `gpeGetMap` using `sbox` as the argument. The value for `map2` is assigned manually.

```
sbox = gpeEditSandbox()
=> pe:#####
map1 = gpeGetMap(sbox)
map2 = list( list("NM0" "A") list("NM1" "B") list("NM2" "C") )
```

In the following example, `sbox` and `map2` values (defined above) are passed as arguments to `gpeSetMap`:

```
gpeSetMap(sbox map2)
=> t
```

In the following example, `sbox` and `map1` values (defined above) are passed as arguments to `gpeSetMap`:

```
gpeSetMap(sbox map1)
=> t
```

gpeSetMapEntry

```
gpeSetMapEntry(
     u_sandbox
     l_mapEntry
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Modifies the map portion of the Modgen sandbox object by updating the symbol mapping value with the specified map entry. The key in the map is the schematic name. If the sandbox has a schematic instance with the same name as the one specified in the map entry, then the symbol mapping of that instance in the sandbox is modified.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
l_mapEntry	Specifies a DPL map entry of the schematic instance name and its symbol mapping.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	Indicates that the Modgen sandbox object was set or modified.
nil	The command was unsuccessful.

Example

Here the two arguments for `gpeSetMapEntry` are first defined. `sb` includes the sandbox object returned by calling `gpeEditSandbox`. `mapEntry` includes the name of the map entry that is created by calling `gpeCreateMapEntry`. These two values are passed as arguments to `gpeSetMapEntry`.

```
sb = gpeEditSandbox()
=> pe:0x#####
mapEntry = gpeCreateMapEntry(?schematicName "M5" ?symbolName "X")
=> (nil symbol "X" schName "M5")
```

```
gpeSetMapEntry(sb mapEntry)
=> t
```

gpeSetMergeLayer

```
gpeSetMergeLayer(
     u_sandbox
     g_mergeLayer { default | well | none | t_layerName | l_layerNames }
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Sets the `MergeLayer` parameter for the specified Modgen sandbox object.

Arguments

u_sandbox Specifies the Modgen sandbox object.

g_mergeLayer

Specifies the layers to be included in the `MergeLayer` parameter. Valid values are:

- `default`: Layer and well merging is done as per the setting of the `modgenMergeWells` and `modgenMergeLayers` environment variables.
- `well`: All well layers in the specified Modgen are merged.
- `none`: No layers and wells are merged.
- *t_layerName*: The shapes on the specified layer are merged. Any DRC violations between the specified layer and other layers of the devices are ignored.
- *l_layerNames*: Merges the specified layers. Before merging, only the top-level net connectivity between the layers is checked. Layers across devices that have different master Pcells can also be merged.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`t`	The `MergeLayer` parameter for the specified Modgen sandbox object has been set.
`nil`	The command was unsuccessful.

Examples

The following examples show the various settings of the *`g_mergeLayer`* argument.

```
sb = gpeEditSandbox()
; Merges layers and wells as per the modgenMergeWells and modgenMergeLayers
environment variables.
gpeSetMergeLayer(sb "default")
; All shapes in the nwell layer are merged.
gpeSetMergeLayer(sb "Nwell")
; Merges the specified layers.
gpeSetMergeLayer(sb list("Nwell" "Oxide"))
=> t
```

gpeStacksAreCompressed

```
gpeStacksAreCompressed(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Determines whether the stacks in the specified Modgen sandbox object are in the compressed state.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	One or more compressed stacks are present in the Modgen sandbox object.
nil	There are no stacks in the Modgen sandbox object.

Example

Determines whether there are any compressed stacks in the specified Modgen sandbox object:

```
lcv = geGetEditCellView()
  mgId = dbGetFigGroupByName(lcv "Modgen_1")
  sb = gpeStartSandbox(?ref mgId)
  gpeStacksAreCompressed(sb)
```

gpeStacksCompress

```
gpeStacksCompress(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Compresses all stacks in the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	All stacks in the specified Modgen sandbox object have been compressed.
nil	There are no stacks in the Modgen sandbox object, or the existing stacks could not be compressed.

Example

Compresses all stacks in the specified Modgen sandbox object sb:

```
lcv = geGetEditCellView()
  mgId = dbGetFigGroupByName(lcv "Modgen_1")
  sb = gpeStartSandbox(?ref mgId)
  gpeStacksCompress(sb)
```

gpeStacksUncompress

```
gpeStacksUncompress(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Uncompresses all stacks in the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	All stacks in the specified Modgen sandbox object have been uncompressed.
nil	There are no stacks in the Modgen sandbox object, or the existing stacks could not be uncompressed.

Example

Uncompresses all stacks in the specified Modgen sandbox object `sb`:

```
lcv = geGetEditCellView()
  mgId = dbGetFigGroupByName(lcv "Modgen_1")
  sb = gpeStartSandbox(?ref mgId)
  gpeStacksUncompress(sb)
```

gpeStartSandbox

```
gpeStartSandbox(
     [ ?ref l_reference ]
     [ ?cv d_cellview ]
     [ ?sync { t | nil } ]
     [ ?updateSchConstraint { t | nil } ]
     [ ?verbose { t | nil } ]
     )
     => g_sandbox / nil
```

Description

Checks for a Modgen sandbox object that includes the specified instances. If one exists, the handle of the Modgen sandbox object is returned for editing with the GPE placement and routing APIs. If an existing object does not have the specified instances, a new Modgen sandbox object is created.

Arguments

`?ref` *`l_reference`*	Specifies a list of instances, including the Modgen figGroups and constraints, to be included in the Modgen sandbox. the default is the current selection.
`?cv` *`d_cellview`*	Specifies the cellview ID. If unspecified, uses the current cellview.
`?sync { t | nil }`	Turns on the synchronization mode for the Modgen sandbox object created. The default is `nil`.
`?updateSchConstraint { t | nil }`	Transfers the constraint to the schematic and the object to the storage. The default is `t`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

g_sandbox	Returns the handle of the Modgen sandbox object that is open for editing.
nil	The Modgen sandbox object could not be created or opened for editing.

Examples

A Modgen sandbox object is created and its handle returned when a list of instances is passed as reference:

```
inst1 = dbFindInstByName("M1")
inst2 = dbFindInstByName("M2")
sbox = gpeStartSandbox(?ref list(inst1 inst2))
```

A Modgen sandbox object is created with the selected instances and its handle returned when the selection made on the canvas contains instances, but not Modgen figGroups:

```
sbox = gpeStartSandbox()
```

A Modgen sandbox object handle is returned for edit when a Modgen figGroup ID is passed as reference:

```
figId = leGetEditFigGroup()
sbox = gpeStartSandbox(?ref figId)
```

A Modgen sandbox object handle is returned for edit when a Modgen constraint ID is passed as reference:

```
mgCiCon = *valid_Modgen_constraint_ID*
sbox = gpeStartSandbox(?ref mgCiCon)
```

A Modgen sandbox object handle is returned for edit when a Modgen constraint is selected in the constraint manager.

```
sbox = gpeStartSandbox()
```

A Modgen sandbox object handle is returned for edit when a list of instances that are part of a Modgen is passed as reference:

```
inst1 = dbFindInstByName("M1")
inst2 = dbFindInstByName("M2")
inst3 = dbFindInstByName("M3")
inst1~>figGroup == inst2~>figGroup == inst3~>figGroup &&
inst1~>figGroup~>type == 'modgen
sbox = gpeStartSandbox(?ref list(inst1 inst2 inst3)
```

A Modgen sandbox object handle is returned for edit when the selection made on the canvas contains Modgen figGroups. The function returns the sandbox handle of the first Modgen figGroup found.

```
sbox = gpeStartSandbox()
```

gpeSyncSandbox

```
gpeSyncSandbox(
     u_sandbox
     [ g_transferConstraint ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Sets the specified Modgen sandbox object to sync mode, meaning that changes made to the state of the Modgen sandbox object cause an update to any associated Modgen constraint and layout placement.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
g_transferConstraint	Specifies whether the constraint must be transferred to the schematic cellview. The default is `t`. Set the argument to `nil` to optimize the script.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Indicates that the specified Modgen sandbox is in the sync mode.
`nil`	The command was unsuccessful.

Example

Sets the Modgen sandbox object, which is identified by the value of `sb`, in sync mode:

```
sb = gpeEditSandbox() => pe:0x#####
gpeSyncSandbox(sb) => t
```

gpeUnAbutGridEntries

```
gpeUnAbutGridEntries(
     u_sandbox
     l_gridEntries
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Unabuts the specified list of instances or indexes in the specified sandbox object.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
l_gridEntries	Includes a list of entries that are specified by indexes. Example: '('(1 1) '(1 2) ...)
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	The instances were unabutted.
nil	The command was unsuccessful.

Example

Makes the Modgen sandbox `sb` editable, and then turns on the synchronization mode. `gpeAbutGridEntires` is used to abut a specified list of instances. `gpeUnAbutGridEntires` is then used to unabut these instances.

```
sb = gpeEditSandbox()
gpeAbutGridEntries(sb list(list(1 0) list(1 1)) ?abutType "type0" ?verbose t)
>t
gpeUnAbutGridEntries(sb list(list(1 0) list(1 1)) ?verbose t)
> t
gpeSyncSandbox(sb)
```

gpeUnSyncSandbox

```
gpeUnSyncSandbox(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Sets the specified Modgen sandbox object to the unsynchronized mode. In this mode, any change to the state of the Modgen sandbox object is not reflected in any associated Modgen constraint and the layout placement.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	Indicates that the state of the Modgen sandbox object has been set to no-sync.
nil	The command was unsuccessful.

Example

Sets the Modgen sandbox object, which is identified by the value of `sb`, to no-sync mode:

```
sb = gpeEditSandbox() => pe:0x#####
gpeUnSyncSandbox(sb) => t
```

gpeAddMatchGroups

```
gpeAddMatchGroups(
     u_sandbox
     l_matchGroups
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Adds the specified matched groups to the specified Modgen sandbox object. A matched group is a set of topological trunks with similar properties.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
l_matchGroups	Specifies a lists of matched group entries created by `gpeCreateMatchGroupEntry`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	The matched groups were added to the Modgen sandbox object.
`nil`	The command was unsuccessful.

Example

In the following example, a matched group `mg1` of match type `widenFirst` is created and then added to a Modgen sandbox object:

```
mg1 = gpeCreateMatchGroupEntry(0 ?twigMatchType "widenFirst")
gpeAddMatchGroups(sandbox list(mg1))
```

gpeAddStrapEntries

```
gpeAddStrapEntries(
     u_sandbox
     l_straps
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Adds the specified straps to the topology (`topo`) section of the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
l_straps	Lists the straps to be added.7
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	Indicates that the straps have been added to the Modgen sandbox object.
nil	The command was unsuccessful.

Example

The following example creates instance terminal entries, and then uses these to create strap entries. The strap entries are then used in `gpeAddStrapEntries` to add straps to a specified Modgen sandbox:

```
strapConstraints = list(
  list("validLayers" nil list("Metal1"))
  list("minWidth" list("Metal1") 0.14)
)
its1 = gpeCreateInstTermEntries(?deviceNames list("M1" "M2") ?instTermNames
list("D" "G"))
```

```
straps1=gpeCreateStrapEntries(?netNames list("vssa!") ?instTermEntries list(its1)
?constraints strapConstraints)
sandbox = gpeEditSandbox(leGetEditFigGroup())
gpeAddStrapEntries(sandbox straps1)
```

gpeAddTrunkChains

```
gpeAddTrunkChains(
     u_sandbox
     l_trunkChains
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Adds the specified trunk chain values to the `topo` portion of the Modgen sandbox object.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
l_trunkChains	Lists the Modgen sandbox `trunkChains` to be added.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Indicates that the trunk chains were added to the Modgen sandbox.
`nil`	Indicates that the command was unsuccessful.

Example

In the following example, `chain1` and `chain2` are trunk chains that store the return values of two separate calls to `gpeCreateTrunkChain`. These values are passed as parameters in `gpeAddTrunkChains`.

```
chain1 = gpeCreateTrunkChain(?trunks trunks1 ?anchorIndex 0 ?trunkDirection
"horizontal" ?offsetDirection "positive" ?anchorPoint "topRight" ?anchorReference
"Metal1")

chain2 = gpeCreateTrunkChain(?trunks trunks2 ?anchorIndex 1 ?trunkDirection
"horizontal" ?offsetDirection "positive" ?anchorPoint "topRight" ?anchorReference
"Metal1")

sandbox=gpeEditSandbox(leGetEditFigGroup())

gpeAddTrunkChains(sandbox list(chain1 chain2))
```

gpeClearChannelWidthEntry

```
gpeClearChannelWidthEntry(
     u_sandbox
     x_index
     t_direction
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Clears the `channelWidth` value of the specified channel in the specified sandbox.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
x_index	Specifies a channel number.
t_direction	Indicates the direction of the `channelWidth` sandbox entry. Valid values are `horizontal` and `vertical`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	The `channelWidth` was deleted.
`nil`	The command was unsuccessful.

Example

In the following example, the `channelWidth` value of the specified channel in the specified sandbox is deleted.

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
gpeClearChannelWidthEntry(sandbox 1 "horizontal")
```

gpeClearMatchGroups

```
gpeClearMatchGroups(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Removes all the matched groups from the specified Modgen sandbox object.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	All the matched groups were removed from the Modgen sandbox object.
nil	The command was unsuccessful.

Example

In the following example, a matched group `mg1` of match type `widenFirst` is created and then added to a Modgen sandbox object. `gpeClearMatchGroups` is then used to remove all the matched groups from the Modgen sandbox object.

```
mg1 = gpeCreateMatchGroupEntry(0 ?twigMatchType "widenFirst")
gpeAddMatchGroups(sandbox list(mg1))
gpeClearMatchGroups(sandbox)
```

gpeClearStrapEntries

```
gpeClearStrapEntries(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Removes all straps from the specified Modgen sandbox.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	All straps were removed from the specified Modgen sandbox.
nil	The command was unsuccessful.

Example

In the following example, all straps are removed from the specified Modgen sandbox.

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
gpeClearStrapEntries(sandbox)
```

gpeClearTopo

```
gpeClearTopo(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Removes all topologies from the specified Modgen sandbox.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	All topologies were removed from the specified Modgen sandbox.
nil	The command was unsuccessful.

Example

In the following example, all topologies are removed from the specified Modgen sandbox:

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
gpeClearTopo(sandbox)
```

gpeClearTrunkChains

```
gpeClearTrunkChains(
     u_sandbox
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Removes all trunk chains from the Modgen sandbox.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

t	All trunk chains were removed from the Modgen sandbox.
nil	The command was unsuccessful.

Example

In the following example, a `sandbox` object is first set up, and then the value is passed as an argument to `gpeClearTrunkChains`:

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
gpeClearTrunkChains(sandbox)
```

gpeCreateChannelWidthEntry

```
gpeCreateChannelWidthEntry(
     [ ?width f_width ]
     [ ?reference t_reference ]
     [ ?verbose { t | nil } ]
     )
     => g_channelWidthEntry / nil
```

Description

Creates a Modgen sandbox channel width entry that can be passed in `gpeSetChannelWidthEntry` to set the channel widths.

Arguments

`?width` *`f_width`*

Specifies the width of the channel.

`?reference` *`t_reference`*

Specifies the reference layer that defines the channel extents. You can specify either the layer name or the layer-purpose pair.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`g_channelWidthEntry`

Returns the ID of the Modgen channel width entry that was created.

`nil` The command was unsuccessful.

Example

Creates a Modgen channel width entry based on the specified width and reference layer values:

```
gpeCreateChannelWidthEntry(?width 0.3 ?reference list("Metal1" "drawing"))
```

gpeCreateInstTermEntries

```
gpeCreateInstTermEntries(
     [ ?deviceNames l_deviceNames ]
     [ ?instTermNames l_instTermNames ]
     [ ?gridEntries  l_gridEntries ]
     [ ?verbose { t | nil } ]
     )
     => l_instTermEntries / nil
```

Description

Creates `instTerms` for all combinations of the two identifiers and returns a list of `instTerms` names that were created. The return value is used as an argument to twig and strap entries to define the connectivity of the topology objects.

Arguments

`?deviceNames` *`l_deviceNames`*

Specifies the instances to connect to.

`?instTermNames` *`l_instTermNames`*

Lists the instance terminals to connect. Use `allTerms` to indicate all `instTerms`.

Example: `list("NM1.1" "NM1.2")`

`?gridEntries` *`l_gridEntries`*

Lists grid entry objects to connect to.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

l_instTermEntries	Returns a list of Modgen sandbox `instTerm` entries. Each entry is a disembodied property list with parameters: `deviceNames` and `instTermNames`. Example: `(nil instTermNames list("S" "D") deviceNames list("NM1.1" "NM1.2") )`
`nil`	The command was unsuccessful.

Example

Creates an instance terminal entry object based on the specified arguments:

```
gpeCreateInstTermEntries(?deviceNames list("M1" "M2") ?instTermNames list("G"
"D"))
```

gpeCreateMatchGroupEntry

```
gpeCreateMatchGroupEntry(
     d_matchGroupID
     [ ?twigMatchType t_twigMatchType ]
     [ ?trunkMatchType t_trunkMatchType ]
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Creates a DPL that represents a matched group. The DPL can then be added to a Modgen sandbox object using the `gpeAddMatchGroups` function.

Arguments

d_matchGroupID Specifies the ID of the matched group for which a matched group entry must be created.

?twigMatchType *t_twigMatchType*

Specifies the criteria based on which twigs must be matched. Valid values are:

- `lengthenOnly`: Matching is done by lengthening the geometries associated with the twigs.
- `widenOnly`: Matching is done by widening the geometries associated with the twigs.
- `lengthenFirst`: Matching is done by first attempting to lengthen the geometries. If that is unsuccessful, then the geometries are widened.
- `widenFirst`: Matching is done by first attempting to widen the geometries. If that is unsuccessful, then the geometries are lengthened.

?trunkMatchType *t_trunkMatchType*

Specifies the criteria based on which trunks must be matched. Valid value is:

- `matchLength`: Matching is done by adjusting the length of the selected trunks such that they are equal.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`t`	A matched group entry was created.
`nil`	The command was unsuccessful.

Example

In the following example, matched group entries `mg1` and `mg2` are created. Both twig and trunk match types are specified for `mg1`, whereas only the twig match type is specified for `mg2`. These matched groups are then passed as parameters for `gpeAddMatchGroups`.

```
mg1 = gpeCreateMatchGroupEntry(0 ?twigMatchType "widenFirst"
                               ?trunkMatchType "matchLength")
mg2 = gpeCreateMatchGroupEntry(1 ?twigMatchType "widenOnly")

gpeAddMatchGroups(sandbox list(mg1 mg2))
```

gpeCreateStrapEntries

```
gpeCreateStrapEntries(
     [ ?netNames l_netNames ]
     [ ?instTermEntries l_instTermEntries ]
     [ ?direction t_direction ]
     [ ?constraints l_constraints ]
     [ ?template l_template ]
     [ ?verbose { t | nil } ]
     )
     => l_straps / nil
```

Description

Creates straps for the specified nets that connect the specified instance terminals.

Arguments

`?netNames` *`l_netNames`*

Specifies the net names for which straps must be created.

`?instTermEntries` *`l_instTermEntries`*

Lists the instance terminals for which straps are to be created.

`?direction` *`t_direction`*

Indicates the direction of the strap.

Valid values are: `horizontal` and `vertical`.

`?constraints` *`l_constraints`*

Lists the strap constraints to be applied.

`?template` *`l_template`*

Specifies the strap entry that serves as the template.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

l_straps	Returns the list of Modgen sandbox straps that were created.
nil	The command was unsuccessful.

Example

In the following example, values for arguments ?constraints and ?instTermEntries are determined.

- strapConstraints stores a DPL of strap definitions.
- its1 contains a list of instTerm entries generated by running gpeCreateInstTermEntries

The values of strapConstraints and its1 are passed as arguments for gpeCreateStrapEntries:

```
strapConstraints= list(
  list("validLayers" nil list("Metal1"))
  list("minWidth" list("Metal1") 0.14)
)
its1 = gpeCreateInstTermEntries(?deviceNames list("M1" "M2") ?instTermNames
list("D" "G"))
straps1 = gpeCreateStrapEntries(?netNames list("vssa!") ?instTermEntries
list(its1) ?constraints strapConstraints)
```

gpeCreateTrunkChain

```
gpeCreateTrunkChain(
     ?trunks l_trunks
     ?anchorIndex n_anchorIndex
     [ ?trunkDirection { horizontal | vertical } ]
     [ ?offsetDirection { positive | negative } ]
     [ ?anchorPoint t_anchorPoint ]
     [ ?anchorReference t_anchorReference ]
     [ ?centerInChannel { t | nil } ]
     [ ?template l_template ]
     [ ?verbose { t | nil } ]
     )
     => g_trunkChainObject / nil
```

Description

Creates trunk chain entries based on the specified identifiers from the sandbox. It is recommended to use only one of the optional arguments. If none of the optional arguments is provided, all trunk chains available in the Modgen are listed. If multiple optional arguments are provided, the return value is the intersection of all the sets.

Arguments

`?trunks l_trunks`

Lists the trunk entries for which trunk chains must be created.

`?anchorIndex n_anchorIndex`

Specifies the row or column number for placing the anchor index.

`?trunkDirection { horizontal | vertical }`

Specifies the trunk direction.

`?offsetDirection { positive | negative }`

Specifies the default trunk offset direction.

`?anchorPoint` *`t_anchorPoint`*

Specifies the location of the anchor point on the anchor instance. Valid values are `topRight`, `bottomLeft`, and `center`.

`?anchorReference` *`t_anchorReference`*

Specifies a reference layer or layer-purpose pair on the anchor on which the trunk chain must start.

`?centerInChannel { t | nil }`

Specifies whether the trunk chain must be centered in the channel indicated by the `?anchorIndex` value. When set to `nil`, the `?offset` value specified in the call to the `gpeCreateTrunkEntries` function, which was used to create the trunks, is used to space the trunk chain from the anchor instance.

`?template` *`l_template`*

Specifies a trunk chain entry, which serves as a template.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`g_trunkChainObject`	Returns the list of Modgen sandbox trunk chains that were created. Each trunk chain is represented as a disembodied property list with anchor properties and list of trunk DPLs.
`nil`	The command was unsuccessful.

Example

Creates a trunk chain based on the specified arguments.

```
gpeCreateTrunkChain(?trunks trunks ?anchorIndex 0 ?trunkDirection "horizontal"
?offsetDirection "positive" ?anchorPoint "topRight" ?anchorReference "Metal1")
```

gpeCreateTrunkEntries

```
gpeCreateTrunkEntries(
     [ ?netNames lt_netNames ]
     [ ?offset f_offset ]
     [ ?longOffsetSource1 t_longOffsetSource1 ]
     [ ?longOffset1 f_longOffset1 ]
     [ ?longOffsetSource2 t_longOffsetSource2 ]
     [ ?longOffset2 f_longOffset2 ]
     [ ?twigs lg_twigs ]
     [ ?twigConstraints lg_twigConstraints ]
     [ ?constraints lg_constraints ]
     [ ?template l_template ]
     [ ?matchGroup l_matchGroupID ]
     [ ?verbose { t | nil } ]
     )
     => l_trunkObjects / nil
```

Description

Creates a list of trunk entries, one for each `netName`. The properties defined by the arguments are applied to all trunks. To define different properties for each trunk, call the function in a `for` loop and set the arguments, as required, for each unique trunk.

Arguments

`?netNames` *`lt_netNames`*

Specifies a list of net names in the Modgen on which trunks must be created. One trunk is created per net.

`?offset` *`f_offset`*

Specifies a list of trunk offsets. The first value is the offset from the trunk to the anchor instance, and is measured from the center of the trunk to the anchor point. All other values are measured from center to center. If there are more nets than offset values, the last offset in the list is used for the remaining trunks.

`?longOffsetSource1` *`t_longOffsetSource1`*

Specifies the reference from which trunk ends must be trimmed on the left or bottom side. Valid values are:

- `figGroupExtents`: Extends the trunk from boundary to boundary (default).
- `twigExtents`: Trims the trunk at the first instance terminal to which its twigs are connect.
- `instanceExtents`: Trims the trunk at the first instance to which its twigs are connected.
- `connExtents`: Trims the trunk at the first instance terminal to which its twigs are connected or the first orthogonal trunk on the same net, whichever trims less.
- `guardRing`: Trims the trunk at the beginning of the guardRing.

`?longOffset1` *`f_longOffset1`*

Specifies the distance (longitudinal offset) from the reference to trim the trunks on the left or bottom side.

`?longOffsetSource2` *`t_longOffsetSource2`*

Specifies the reference from which trunk ends must be trimmed on the right or top side. Valid values are:

- `figGroupExtents`: Extends the trunk from boundary to boundary (default).
- `twigExtents`: Trims the trunk at the last instance terminal to which its twigs are connect.
- `instanceExtents`: Trims the trunk at the last instance to which its twigs are connected.
- `connExtents`: Trims the trunk at the last instance terminal to which its twigs are connected or the last orthogonal trunk on the same net, whichever trims less.
- `guardRing`: Trims the trunk at the end of the guardRing.

`?longOffset2` *`f_longOffset2`*

Sets the longitudinal offset on the top or right.

`?twigs` *`lg_twigs`*

Lists of twigs to be added to the trunk. This value is usually derived from `gpeCreateTwigs`, which makes it appear such that `gpeSetTwigs` does not exist.

Specify `nil` to clear all twigs.

`?twigConstraints` *`lg_twigConstraints`*

Specifies the constraints to set on the `trunkTwig` constraint group. These constraints apply to the twigs on the trunk.

`?constraints` *`lg_constraints`*

Specifies the constraints to set on the default constraint group. These constraints apply to the trunk.

`?template` *`l_template`*

Specifies the values for parameters that are not set by the function.

`?matchGroup` *`l_matchGroupID`*

Specifies the IDs of the matched groups to which these trunks must be added.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`l_trunkObjects`	Returns a list of Modgen sandbox trunk objects for each net.
`nil`	The command was unsuccessful.

Example

Creates trunk entries based on the specified arguments:

```
trunks=gpeCreateTrunkEntries(?netNames list("vssa!" "vdda!") ?offset 0.4
?longOffset1 0.2 ?longOffsetSource1 "twigExtents" ?longOffset2 0.3
?longOffsetSource2 "instanceExtents")
```

gpeCreateTwigEntries

```
gpeCreateTwigEntries(
     [ ?instTermEntries l_instTermEntries ]
     [ ?constraints l_constraints ]
     [ ?template l_template ]
     [ ?verbose { t | nil } ]
     )
     => g_twigObject / nil
```

Description

Creates a Modgen sandbox twig object, which specifies the instance terminals to make connections to and the constraints that define how the connections must be made. The twig objects can be added to trunk with the `?twigEntries` parameter on `gpeCreateTrunkEntries`.

Arguments

`?instTermEntries` *`l_instTermEntries`*

List of DPLs that specify the instances with the instance terminals to which the twigs must connect. This is typically created from a call to `gpeCreateInstTermEntries`.

`?constraints` *`l_constraints`*

Lists the twig constraints to be set on the default constraint group. Each routing entry comprises a twig constraint name and the arguments for the constraint.

`?template` *`l_template`*

Specifies the parameters if they are otherwise unspecified. Here, it specifies the twig entry.

`?verbose { t | nil }`

Controls the display of warning messages.

Value Returned

`g_twigObject`	Returns the name of a Modgen sandbox twig object.
`nil`	The command was unsuccessful.

Example

In the following example, values for `twigConstraints` and `iterm` are first instantiated. These values are then passed as arguments in `gpeCreateTwigEntries`:

```
twigConstraints = list(
  list("validLayers" nil list("Metal1"))
  list("minWidth" list("Metal1") 0.14)
)


iterm = gpeCreateInstTermEntries(?deviceNames list("M1" "M2") ?instTermNames
list("G" "D"))


twigEntry = gpeCreateTwigEntries(?instTermEntries list(iterm) ?constraints
twigConstraints)
```

gpeGetChannelWidthEntry

```
gpeGetChannelWidthEntry(
     u_sandbox
     x_channelNumber
     t_direction
     [ ?verbose { t | nil } ]
     )
     => g_ChannelWidthEntry / nil
```

Description

Returns the `channelWidth` sandbox object that matches the specified index and direction values.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
x_channelNumber	Specifies the row or column number in which the `channelWidth` sandbox object is located.
t_direction	Specifies the direction of the `channelWidth` sandbox object. Valid values are: `horizontal` and `vertical`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

g_ChannelWidthEntry

Returns a Modgen channel width object that has properties: `width` and `reference`.

Example:

```
(nil 0.8 "Active")
```

`nil` The command was unsuccessful.

Example

Retrieves the channel width from the first horizontal channel in the specified Modgen sandbox object.

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
channelWidth = gpeGetChannelWidthEntry(sandbox 1 "horizontal")
```

gpeGetStrapEntries

```
gpeGetStrapEntries(
     u_sandboxObject
     [ ?netNames l_netNames ]
     [ ?deviceNames l_deviceNames ]
     [ ?instTermNames l_instTermNames ]
     [ ?verbose { t | nil } ]
     )
     => l_straps / nil
```

Description

Filters straps in the Modgen sandbox based on the specified identifiers. It is recommended to use only one of the optional arguments. If no arguments are provided, all straps in the Modgen are displayed. If multiple arguments are provided, the intersection of the sets is displayed.

Arguments

u_sandboxObject	Specifies a Modgen sandbox object.
?netNames *l_netNames*	Lists names of nets, based on which straps must be filtered.
?deviceNames *l_deviceNames*	Lists devices, such as instances and guard rings, based on which straps must be filtered.
?instTermNames *l_instTermNames*	Specifies `instTerm` names based on which straps must be filtered.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_straps	Returns a list of Modgen sandbox straps.
nil	The command was unsuccessful.

Example

Returns all straps with nets `vdda!` or `vssa!` in the specified Modgen sandbox object that are connected to instTerm `G` or `D` on devices `M1` or `M2`:

```
sandbox = gpeEditSandbox(leGetEditFigGroup())

straps = gpeGetStrapEntries(sandbox ?netNames list("vdda!" "vssa!") ?deviceNames
list("M1" "M2") ?instTermNames list("G" "D"))
```

gpeGetTrunkChains

```
gpeGetTrunkChains(
     u_sandbox
     [ ?anchorIndex n_anchorIndex ]
     [ ?trunkDirection t_trunkDirection ]
     [ ?offsetDirection t_offsetDirection ]
     [ ?anchorPoint t_anchorPoint ]
     [ ?verbose { t | nil } ]
     )
     => l_trunksChains / nil
```

Description

Returns a list of trunk chains based on the specified identifiers. If none of the optional arguments are provided, all `trunkChains` available in the Modgen are listed. If multiple optional arguments are provided, the return value is the intersection of all the sets.

Arguments

u_sandbox

Specifies the Modgen sandbox object to query.

?anchorIndex *n_anchorIndex*

Specifies the row or column number for the anchor index.

?trunkDirection *t_trunkDirection*

Specifies the trunk direction. Valid values are `vertical` and `horizontal`.

?offsetDirection *t_offsetDirection*

Specifies the direction in which the offset is measured. Valid values are `positive` and `negative`.

?anchorPoint *t_anchorpoint*

Indicates the location of the anchor point. Valid values are `topRight` and `bottomLeft`.

?verbose { t | nil }

Controls the display of warning messages.

Value Returned

l_trunksChains	Returns a list of trunk chains. Each trunk chain is represented as a disembodied property list with anchor properties and list of trunk DPLs.
`nil`	The command was unsuccessful.

Example

Creates a trunk chain based on the specified arguments:

```
gpeGetTrunkChains(sandbox ?anchorIndex 1)
```

gpeGetTwigEntries

```
gpeGetTwigEntries(
     u_sandbox
     [ ?trunks l_trunks ]
     [ ?verbose { t | nil } ]
     )
     => l_Twigs / nil
```

Description

Returns all twigs on the specified trunks from the specified sandbox.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
?trunks *l_trunks*	Lists the trunks from which twigs must be retrieved.
?verbose { t | nil }	Controls the display of warning messages.

Value Returned

l_Twigs	Returns a list of Modgen sandbox twigs. Each twig is represented as a disembodied property list with the following optional parameters: netName, deviceNames, instTermNames, directions, constraints. Example: `(nil instTermNames list("S" "D") deviceNames list("NM1.1" "NM1.2") constraints list(list("minNumCuts" "Poly" 2))`
nil	The command was unsuccessful.

Example

In the following example, values for sandbox and chain are first determined; these values are then passed as arguments in gpeGetTwigEntries:

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
chain = car(sandbox->topo->trunkChains)
```

```
twigs = gpeGetTwigEntries(sandbox chain->trunks)
```

gpeSetChannelWidthEntry

```
gpeSetChannelWidthEntry(
     u_sandbox
     x_index
     t_direction
     g_channelWidth
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Sets the `channelWidth` for the specified channel.

Arguments

u_sandbox	Specifies a Modgen sandbox object.
x_index	Specifies the channel number.
t_direction	Specifies the direction of the `channelWidth` sandbox object. Valid values are `horizontal` and `vertical`.
g_channelWidth	Specifies the channel width to be set, typically created by a call to `gpeCreateChannelWidthEntry`.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	The `channelWidth` was set.
`nil`	The command was unsuccessful.

Example

In the following example, `gpeCreateChannelWidthEntry` is run, and its return values are stored in `cw`. `gpeEditSandbox` is run and its return values are stored in `sandbox`. `cw` and `sandbox` values are used as arguments in `gpeChannelWidthEntry`:

```
cw = gpeCreateChannelWidthEntry(?width 0.3 ?reference list("Metal1" "drawing"))
sandbox = gpeEditSandbox(leGetEditFigGroup())
```

```
gpeSetChannelWidthEntry(sandbox 2 "horizontal" cw)
```

gpeSetRouter

```
gpeSetRouter(
     u_sandbox
     t_router
     [ ?verbose { t | nil } ]
     )
     => t / nil
```

Description

Specifies the router to be used to route the Modgen.

Arguments

u_sandbox	Specifies the Modgen sandbox object.
t_router	Specifies the name of the router to be set. Valid values are: ■ `pinToTrunk`: Calls the `PinToTrunk` router, with the topology information specified in the Modgen to do the routing. ■ `none`: The Modgen is not routed by any router. ■ `manual`: The current routing in the Modgen is not changed. The Modgen can be routed manually by drawing the physical geometry inside it.
`?verbose { t | nil }`	Controls the display of warning messages.

Value Returned

`t`	Indicates that the router parameter was set.
`nil`	The command was unsuccessful.

Example

In the following example, the pin to trunk router is set to route the specified Modgen sandbox object.

```
sandbox = gpeEditSandbox(leGetEditFigGroup())
gpeSetRouter(sandbox "pinToTrunk")
=> t
```

9

Abstract Generator SKILL Functions

This topic contains information about the SKILL functions used in Abstract Generator.You can use the same functions in TCL mode by typing `absTclMode` in the *Command Entry Field*. Note that TCL does not accept parentheses.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Entering Functions

When entering SKILL functions in Abstract Generator, the processes of entering parameters and options are different.

- Set parameters directly on the function line; for example,

```
absSetLibrary "libname"
```

- Set options prior to calling the function, using either the `absSetOption` or `absSetBinOption` functions; for example,

```
absSetOption "ImportLogicalFiles" "logic1"
absSetOption "ImportLogicalType" "Verilog"
absImportLogical
```

Completing Functions

If you type in part of a function and press the tab key, Abstract Generator tries to complete the function. If it is unable to do so, it presents a list of possible function names that begin with the partial string you entered.

For example, because all of the SKILL API functions begin with the string `abs`, if you type in `a` and press *Tab*, you will see a list of ALL the function names. Function completion works only for functions beginning with `abs`.

In Tcl mode only, if you need to know the parameters of a function quickly, type the function name and press *Return*. If the function requires arguments, Abstract Generator lists them. If there are no required arguments, the function is executed.

For example,

```
abstract> absSetLibrary
Wrong # args. absSetLibrary name
```

The `absSetLibrary` function requires the library name to be supplied.

Record File

To find out more about SKILL functions, examine the `abstract.record` file, which contains the record of a session expressed in SKILL functions.

To find a particular SKILL function, use the graphical user interface to perform the action (for example, setting a bin option), then close the program and examine the entries in the `abstract.record` file.

Classification of Abstract Generator SKILL Functions

The SKILL functions that let you manage the tasks in the Virtuoso® Abstract Generator can be broadly classified in the following categories.

- Importing Functions: Use the following functions to import data into Abstract Generator.
  - absImportGDS
  - absImportOasis
  - absImportLogical
  - absImportLEF
  - absImportDEF
  - absImportCTLF
  - absImportVerilog
  - absImportOptions
- Exporting Functions: Use the following functions to export data from Abstract Generator.
  - absExportLEF
  - absExportReport
  - absExportOptions
- Library Functions: Use the following functions to set and get library names and attach technology information.

- absGetLibrary
- absSetLibrary
- absAttachTechLib

- Selecting Functions: Use the following functions to select objects in Abstract Generator.
  - absSelect
  - absSelectAllBins
  - absSelectCells
  - absSelectBin
  - absSelectBinFrom
  - absSelectCell
  - absGetSelectedCells
  - absSelectCellFrom
- Deselecting Functions: Use the following functions to deselect objects in Abstract Generator.
  - absDeselectAllBins
  - absDeselectCells
  - absDeselectBin
  - absDeselectBinFrom
  - absDeselectCell
  - absDeselectCellFrom
  - absDeselectCellsInList
- Bin Operation Functions: Use the following functions to manipulate bins in Abstract Generator:
  - absGetBins
  - absGetBinType
  - absGetSelectedBins
  - absMoveSelectedCellsToBin

- absRenameBin
- absNewBin
- absDeleteBin
- absDeleteBinMoveCellsTo
- absCopyBinOptions

- Option Functions: Use the following functions to get and set global options in Abstract Generator.
  - absGetOption
  - absSetOption
- Bin Option Functions: Use the following functions to set and get bin options in Abstract Generator.
  - absGetBinOption
  - absSetBinOption
- Flow Functions: Use the following functions to execute flow steps in Abstract Generator.
  - absPins
  - absExtract
  - absAbstract
  - absVerify
- Cell Property Functions: Use the following functions to set and get cell properties in Abstract Generator.
  - absGetCellProp
  - absSetCellProp
- Terminal Property Functions: Use the following functions to set and get terminal properties in Abstract Generator.
  - absGetTerminalProp
  - absSetTerminalProp
- Miscellaneous Functions: The following SKILL functions do not fit into any of the other categories.
  - absDisableUpdate

- absDistributeCells
- absEnableUpdate
- absExit
- absRevalidateSelectedCells
- absSort
- absVersion
- iagCreateMenuItem
- iagGenAbstract

absImportGDS

```
absImportGDS(
     )
     => 0 / 1
```

Description

Creates cellviews from GDSII layout data.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportGDSIIFiles` *`filenames`*

Specifies one or more GDSII stream file names that you want to import into Abstract Generator.

`ImportGDSIICaseMap { No Mapping | to Uppercase | to Lowercase }`

Maps the cell names in GDSII stream files to lower or upper case or specifies that cell names are not to be mapped.

`ImportGDSIITemplateFile` *`filename`*

Specifyies one stream `.template` filename at a time to be imported into Abstract Generator.

`DefaultBin { Core | IO | Corner | Block | Ignore }`

Selects the bin that you want library cells to be imported to. The default setting is `Core`.

Examples

Imports a GDSII file called `stream1`, maps the cell names to upper case, and moves them into the `Core` bin. The template file `stream1.template` is imported at the same time.

```
absSetOption "ImportGDSIIFiles" "stream1"
=> 1
absSetOption "ImportGDSIICaseMap" "to Uppercase"
=> 1
absSetOption "ImportGDSIITemplateFile" "stream1.template"
=> 1
absSetOption "DefaultBin" "Core"
=> 1
absImportGDS
=> 1
```

Related Topics

Importing GDSII Information in Abstract Generator

absImportOasis

```
absImportOasis(
     )
     => 0 / 1
```

Description

Creates cellviews from OASIS data.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportOasisFiles` *`OasisFilenames`*

Specifies the OASIS files to import.

`OasisImportTechRefLib` *`libName`*

Specifies the reference technology library to be used while importing the OASIS files.

`ImportOasisCaseMap { No Mapping | to Uppercase | to Lowercase }`

Specifies the casing to be followed while mapping cell names in the OASIS files: uppercase or lowercase. Also specifies if the cell names must not be mapped.

`ImportOasisTemplateFile` *`filename`*

Specifies a file that has the `.template` extension to be used during import.

`DefaultBin { Core | IO | Corner | Block | Ignore }`

Specifies the bin in which the imported library cells must be placed. The default value is `Core`.

`ImportOasisEnableColoring { true | false }`

Specifies whether coloring must be enabled when importing the OASIS files.

`ImportOasisLayerMapFile` *`layerMapFileName`*

Specifies the layer map file to be used.

Examples

Imports an OASIS file named `file1`, maps the cell names to upper case, and moves them into the `Core` bin. Coloring is enabled and a layer map file is specified. The template file `file1.template` is imported at the same time.

```
absSetOption( "ImportOasisFiles" "file1" )
=> 1
absSetOption( "ImportOasisLayerMapFile" "layerMapfile1" )
=>1
absSetOption( "ImportOasisCaseMap" "to Uppercase" )
=> 1
absSetOption( "ImportOasisTemplateFile" "file1.template" )
=> 1
absSetOption( "DefaultBin" "Core" )
=> 1
absSetOption( "ImportOasisEnableColoring" "true" )
=>1
absImportOasis
=> 1
```

Related Topics

Importing OASIS Data in Abstract Generator

absImportLogical

```
absImportLogical(
     )
     => 0 / 1
```

Description

Creates cellviews from existing logical information, typically represented in LIB or Verilog formats.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportLogicalFiles` *`filenames`*

Specifies the names of one or more logical files to import.

`ImportLogicalType { LIB | Verilog }`

Specifies the logical file type format, `LIB` or `Verilog`. The default setting is `LIB`.

Examples

Imports a Verilog logical file called `logic1`.

```
absSetOption( "ImportLogicalFiles" "logic1" )
=> 1
absSetOption( "ImportLogicalType" "Verilog" )
=> 1
absImportLogical
=> 1
```

Related Topics

Importing Logical Data in Abstract Generator

absImportLEF

```
absImportLEF(
     )
     => 0 / 1
```

Description

Creates cellviews from the LEF data.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportLefFiles` *`filenames`*

Specifies the LEF files to import.

`ImportLefView` *`viewname`*

Specifies the cellview that a LEF macro will be saved to.

`ImportTechLef { true | false }`

Control whether LEF macros are overwritten or appended to when antenna LEF is imported after geometry LEF. The default setting is `false`.

Examples

Imports a LEF file called `myleffile1` and specifies that LEF macro information be saved to the `abstract` view without overwriting any existing LEF data.

```
absSetOption( "ImportLefFiles" "myleffile1" )
=> 1
absSetOption( "ImportLefView" "abstract" )
=> 1
```

```
absSetOption( "ImportTechLef" "false" )
=> 1
absImportLEF
=> 1
```

Related Topics

Importing LEF in Abstract Generator

absImportDEF

```
absImportDEF(
     )
     => 0 / 1
```

Description

Creates cellviews from DEF data.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportDefFiles` *`filenames`*

Specifies the name of the DEF files to be imported.

`DefaultBin { Core | IO | Corner | Block | Ignore }`

Selects the bin that you want library cells to be imported to. The default setting is `Core`.

Examples

Imports a DEF file called `mydeffile1`and moves the views into the `Core` bin.

```
absSetOption( "ImportDefFiles" "mydeffile1" )
=> 1
absSetOption( "DefaultBin" "Core" )
=> 1
absImportDEF
=> 1
```

Related Topics

Importing DEF in Abstract Generator

absImportCTLF

```
absImportCTLF(
     )
     => 0 / 1
```

Description

Creates cellviews from CTLF logical data.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportCTLFFiles` *`filenames`*

Specifies the names of the CTLF files to be imported.

Examples

```
absSetOption "ImportCTLFFiles" "myctlffile1"
=> 1
absImportCTLF
=> 1
```

Imports a CTLF file called `myctlffile1`.

Related Topics

Importing LEF in Abstract Generator

absImportVerilog

```
absImportVerilog(
     )
     => 0 / 1
```

Description

Creates cellviews from Verilog logical data.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`ImportVerilogFiles` *`filenames`*

Specifies the name of the CTLF file to be imported.

Examples

Imports a Verilog file called `myverilogfile1`.

```
absSetOption "ImportVerilogFiles" "myverilogfile1"
=> 1
absImportVerilog
=> 1
```

Related Topics

Import Data in Abstract Generator

absImportOptions

```
absImportOptions(
     )
     => 0 / 1
```

Description

Loads a previous session's saved options file from the specified location.

Arguments

None

Value Returned

`0`	An error is encountered.
`1`	The function runs successfully.

Options

`ImportOptionsFile` *`filename`*

Specifies the path and name of the options file to be loaded. The default is `abstract.options`.

Examples

Imports the options from the file called `abstract.options` saved in the current working directory.

```
absSetOption "ImportOptionsFile" "abstract.options"
=> 1
absImportOptions
=> 1
```

Related Topics

Import Data in Abstract Generator

absExportLEF

```
absExportLEF(
     )
     => 0 / 1
```

Description

Generates LEF files for cells in one or more bins. All exported LEF data is contained in the LEF *filename*.

Arguments

None

Value Returned

`0`	An error is encountered.
`1`	The function executes successfully.

Options

`ExportLEFFile` *filename*

Specifies the path to and name of the LEF file to export.

`ExportGeometryLefData { true | false }`

Controls whether geometry LEF is exported. The default setting is `true`.

`ExportTechLefData { true | false }`

Specifies whether technology LEF is exported. The default setting is `true`.

`ExportLEFVersion { 5.3 | 5.4 | 5.5 | 5.6 | 5.7}`

Specifies the version of LEF to export. The default setting value is `5.4`, if the `sedsm` is in your path.

`ExportLEFBin { All | Core | IO | Corner | Block | Ignore }`

Specifies the bins that contains the cells for which you want LEF generated. The default setting is `All`.

Examples

Exports a LEF 5.5 compatible file called `file1.lef` containing LEF geometry and technology information for all the cells contained in all the bins.

```
absSetOption( "ExportLEFFile" "file1.lef" )
=> 1
absSetOption( "ExportLEFVersion" "5.5" )
=> 1
absSetOption( "ExportGeometryLefData" "true" )
=> 1
absSetOption( "ExportTechLefData" "true" )
=> 1
absSetOption( "ExportLEFBin" "All" )
=> 1
absExportLEF
=> 1
```

Related Topics

Exporting LEF in Abstract Generator

absExportReport

```
absExportReport(
     )
     => 0 / 1
```

Description

Outputs a summary of the cells in your library to a report file.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Option

`ExportReportFile` *`filename`*

Specifies the path to and name of the report file. The default is `abstract.report`.

Examples

Exports a report file called `file1.rep` in the current working directory.

```
absSetOption( "ExportReportFile" "file1.rep" )
=> 1
absExportReport
=> 1
```

Related Topics

Exporting a Report in Abstract Generator

absExportOptions

```
absExportOptions(
     )
     => 0 / 1
```

Description

Save the options settings for the current session to a specified location.

Arguments

None

Value Returned

`0`	An error is encountered.
`1`	The function executes successfully.

Option

`ExportOptionsFile` *`filename`*

Specifies the path to and name of the options file. The default is `abstract.options`.

Examples

Saves the options in a file called `abstract.options` in the current working directory.

```
absSetOption( "ExportOptionsFile" "abstract.options" )
=> 1
absExportOptions
=> 1
```

Related Topics

Exporting a Report in Abstract Generator

absGetLibrary

```
absGetLibrary(
     )
     => library_name
```

Description

Returns the name of the current library.

Arguments

None

Value Returned

library_name	Name of the library loaded in the current session of abstract generator.

Related Topics

Creating a Library in Abstract Generator

absSetLibrary

```
absSetLibrary(
     name
     )
     => t / nil
```

Description

Specifies the library that you want to process through Abstract Generator. This involves either creating a new library and setting into it or setting into an existing library.

Arguments

name	Specifies the name of the library to be processed. **Note:** If you specify a library name that does not already exist, Abstract Generator creates a new library of that name.

Value Returned

`t`	The function is executed successfully.
`nil`	An error is encountered.

Example

Sets the current library in Abstract Generator to be `MyLibrary`.

```
absSetLibrary( "MyLibrary" )
=> t
```

Related Topics

Creating a Library in Abstract Generator

absAttachTechLib

```
absAttachTechLib(
     name
     )
     => t / nil
```

Description

Attaches the named technology library to the current library.

Arguments

name	Specifies the name of the technology library to be attached. You must set the current library before attaching a technology library.

Value Returned

t	The function is executed successfully.
nil	An error is encountered.

Examples

Attaches the technology library, techlib1, to the current library, MyLibrary.

```
absSetLibrary( "MyLibrary" )
=> t
absAttachTechLib( "techlib1" )
=> t
```

Related Topics

Attaching Technology Information to a Library in Abstract Generator

absSelect

```
absSelect(
     )
     => 0 / 1
```

Description

Selects cells according to the specified options.

Arguments

None

Value Returned

`0`	Indicates an error.
`1`	Indicates successful operation.

Options

`SelectInvalid { true | false }`

Specifies whether or not to select cells with an invalid status. The default setting is `false`.

`SelectValid { true | false }`

Specifies whether or not to select cells with a valid status. The default setting is `false`.

`SelectPropertyVal` *`property_value`*

Selects only cells that have a particular property value.

`SelectPropertyName` *`property_name`*

Selects only cells that have the named property as an attribute.

`SelectHeightTo` *`max_height`*

Selects only cells that do not exceed the specified height.

`SelectHeightFrom` *`min_height`*

Selects only cells that exceed the specified height.

`SelectView { All | Layout | Logical | Pins | Extract | Abstract | Verify }`

Selects only cells that have the specified view. The default setting is `All`.

`SelectName` *`regular_expression`*

Selects only cells with names matching the specified regular expression.

`SelectError { true | false }`

Specifies whether or not to select cells with status errors.

`SelectByMsg` *`string`*

Selects only cells with errors matching the specified string.

Examples

Selects all cells with a valid abstract view.

```
absSetOption( "SelectValid" "true" )
=> 1
absSetOption( "SelectView" "Abstract" )
=> 1
absSelect
=> 1
```

Related Topics

Selecting Cells Based on a Criteria in Abstract Generator

absSelectAllBins

```
absSelectAllBins(
     )
     => integer
```

Description

Selects all the bins in the current library.

Arguments

None

Value Returned

integer	Returns the number of bins that Abstract Generator selects.

Related Topics

Moving Cells Across Bins in Abstract Generator

absSelectCells

```
absSelectCells(
     )
     => integer
```

Description

Selects all cells in the selected bins.

Arguments

None

Value Returned

integer	Returns the number of cells that Abstract Generator selects.

Related Topics

Selecting Cells Based on a Criteria in Abstract Generator

absSelectBin

```
absSelectBin(
     name
     )
     => 0 / 1
```

Description

Selects the named bin. This makes the specified bin current for use with other functions.

Arguments

name	Specifies the name of the bin to select.

Value Returned

0	An error is encountered.
1	The function executes successfully.

Example

Selects the `Core` bin and makes it the current bin.

```
absSelectBin( "Core" )
=> 1
```

Related Topics

Moving Cells Across Bins in Abstract Generator

absSelectBinFrom

```
absSelectBinFrom(
     from
     to
     )
     => integer
```

Description

Selects a range of bins. The selection is made from top to bottom starting from the *from* bin. The *from* bin should be above the *to* bin the Bins pane, otherwise all bins listed after *from* are selected.

Arguments

from	Specifies the first bin to be selected in the range.
to	Specifies the last bin to be selected in the range.

Value Returned

integer	Returns the number of bins that Abstract Generator selects.

Example

Selects all bins in the range of `userbin1` to `userbin9` and returns the number of bins selected.

```
absSelectBinFrom( "userbin1" userbin9" )
=> 2
```

Related Topics

Moving Cells Across Bins in Abstract Generator

absSelectCell

```
absSelectCell(
     name
     )
     => 0 / 1
```

Description

Selects a single named cell.

Arguments

name	Specifies the name of the cell to be selected.

Value Returned

0	An error is encountered.
1	The function executes successfully.

Example

Selects `cell1`.

```
absSelectCell "cell1"
=> 1
```

Related Topics

Selecting Cells Based on a Criteria in Abstract Generator

absGetSelectedCells

```
absGetSelectedCells(
     )
     => string
```

Description

Returns the list of cells selected in the *Cell* pane. The cells might belong to different bins.

Arguments

None

Value Returned

string	Returns the list of selected cells.

Example

Returns the currently selected cells in the *Cell* pane — `cell1`, `BUF2`, and `pll`.

```
absGetSelectedCells
=> List of selected cells : cell1 BUF2 pll
```

Related Topics

Selecting Cells Based on a Criteria in Abstract Generator

absSelectCellFrom

```
absSelectCellFrom(
     from
     to
     )
     => integer
```

Description

Selects a range of cells. The selection is made from top to bottom starting from the *from* cell. The *from* cell should be above the *to* cell the *Cell* pane, otherwise all cells listed after *from* are selected.

Arguments

from	Specifies the first cell to be selected in the range.
to	Specifies the last cell to be selected in the range.

Value Returned

integer	Returns the number of cells that Abstract Generator selects.

Example

Selects all cells in the range from `ABC1` through `ABC9` and returns the number of cells selected.

```
absSelectCellFrom "ABC1" "ABC9"
=> 2
```

Related Topics

Selecting Cells Based on a Criteria in Abstract Generator

absSelectCellsInList

```
absSelectCellsInList(
     cellNameList
     )
     => integer
```

Description

Selects a list of cells that are not adjacent to each other in the *Cell* pane.

Arguments

cellNameList	Specifies a space-separated list of cell names to be selected.

Value Returned

integer	Returns the number of cells that Abstract Generator selects.

Examples

Selects the cells `cell1`, `cell2`, `cell6`, and `cell9`, makes them current in Abstract Generator, and returns `4` as the number of the cells that Abstract Generator selects.

```
absSelectCellsInList "cell1 cell2 cell6 cell9"
=> 4
```

Related Topics

Selecting Cells Based on a Criteria in Abstract Generator

absDeselectAllBins

```
absDeselectAllBins(
     )
     => integer
```

Description

Deselects all currently selected bins.

Arguments

None

Value Returned

integer	Returns the number of cells that Abstract Generator selects.

Related Topics

Cell Preferences for Abstract Generation

absDeselectCells

```
absDeselectCells(
     )
     => integer
```

Description

Deselects all cells in the current bins.

Arguments

None

Value Returned

integer	Returns the number of cells that Abstract Generator deselects.

Related Topics

Cell Preferences for Abstract Generation

absDeselectBin

```
absDeselectBin(
     name
     )
     => 0 / 1
```

Description

Deselects a named bin.

Arguments

name	Specifies the name of the bin to deselect.

Value Returned

0	Indicates an error.
1	Indicates successful operation.

Examples

```
absDeselectBin "Core"
=> 1
```

Deselects the `Core` bin.

```
absDeselectBin "Core"
=> 0
```

The `Core` bin is already deselected.

Related Topics

Cell Preferences for Abstract Generation

absDeselectBinFrom

```
absDeselectBinFrom(
     from
     to
     )
     => integer
```

Description

Deselects a range of bins. The deselection is done from top to bottom starting from the *from* bin. The *from* bin should be above the *to* bin the *Bin* pane, otherwise all bins listed after *from* are deselected.

Arguments

from	Specifies the first bin to be deselected in the range.
to	Specifies the last bin to be deselected in the range.

Value Returned

integer	Returns the number of bins that Abstract Generator deselects.

Examples

```
absDeselectBinFrom "userbin1" "userbin9"
=> 3
```

Deselects all bins in the range of `userbin1` to `userbin9`.

Related Topics

Cell Preferences for Abstract Generation

absDeselectCell

```
absDeselectCell(
     name
     )
     => 0 / 1
```

Description

Deselects a named cell.

Arguments

name	Specifies the name of the cell to be deselected.

Value Returned

0	Indicates an error.
1	Indicates successful operation.

Examples

Deselects `cell1`.

```
absDeselectCell "cell1"
=> 1
```

Related Topics

Cell Preferences for Abstract Generation

absDeselectCellFrom

```
absDeselectCellFrom(
     from
     to
     )
     => integer
```

Description

Deselects a range of cells. The deselection is done from top to bottom starting from the *from* cell. The *from* cell should be above the *to* cell the *Cell* pane, otherwise all cells listed after *from* are deselected.

Arguments

from	Specifies the first cell to be deselected in the range.
to	Specifies the last cell to be deselected in the range.

Value Returned

integer	Returns the number of cells that Abstract Generator deselects.

Example

Deselects all cells in the range of ABC1 to ABC9.

```
absDeselectCellFrom "ABC1" "ABC9"
=> 2
```

Related Topics

Cell Preferences for Abstract Generation

absDeselectCellsInList

```
absDeselectCellsInList(
     cellNameList
     )
     => integer
```

Description

Deselects a list of cells that are not adjacent to each other in the *Cell* pane.

Arguments

cellNameList	Specifies a space-separated list of cell names to be deselected.

Value Returned

integer	Returns the number of cells that Abstract Generator deselects.

Example

Deselects `cell1`, `cell2`, `cell6` and `cell9`.

```
absDeselectCellsInList "cell1 cell2 cell6 cell9"
=> 4
```

Related Topics

Cell Preferences for Abstract Generation

absGetBins

```
absGetBins(
     )
     => string
```

Description

Displays the list of available bins. The list returned includes the five system bins as well as the user bins, if any.

Arguments

None

Value Returned

string	Returns the list of available bins.

Examples

Displays all the system bins.

```
absGetBins
=> List of bins : Core IO Corner Block Ignore
```

Displays the five system bins and the user bin `myBin`.

```
absGetBins
=> List of bins : Core IO Corner Block Ignore myBin
```

Related Topics

Cell Preferences for Abstract Generation

absGetBinType

```
absGetBinType(
     name
     )
     => string
```

Description

Returns the type of the bin passed as an argument. Both system and user bins can be specified as arguments.

The bin types of the system bins are specified below.

Core --> standard
IO --> IO
Corner --> corner
Block --> macro
Ignore --> none

Arguments

name Specifies the name of the bin whose type is to be determined.

Value Returned

string Returns the type of the specified bin.

Examples

Displays the type of the bin `IO`.

```
absGetBinType "IO"
=> Bin type of bin 'IO' : IO
```

Displays the type of the `Block` bin.

```
absGetBinType "Block"
=> Bin type of bin 'Block' : macro
```

Displays the type of the user bin `myBin` as `corner`.

```
absGetBinType "myBin"
=> Bin type of bin 'myBin' : corner
```

Related Topics

Bin Preferences for Abstract Generation

absGetSelectedBins

```
absGetSelectedBins(
     )
     => string
```

Description

Returns the list of bins selected in the *Bin* pane. The order of the bins in the list matches the order in which they appear in the *Bin* pane.

Arguments

None

Value Returned

string	Returns the list of the selected bins.

Examples

Returns `Core` as the bin selected in the *Bin* pane.

```
absGetSelectedBins
=> List of selected bins : Core
```

Returns the list `IO Corner Ignore` as the bins selected in the *Bin* pane.

```
absGetSelectedBins
=> List of selected bins : IO Corner Ignore
```

Related Topics

Bin Preferences for Abstract Generation

absMoveSelectedCellsToBin

```
absMoveSelectedCellsToBin(
     name
     )
     => integer
```

Description

Moves any currently selected cells to a named bin. This can be one of the five system bins or a user-defined bin.

Arguments

name Specifies the name of the bin to which the selected cell(s) are to be moved.

Value Returned

integer Returns the number of cells moved.

Example

Moves the selected cells to the IO bin and returns the number of cells moved, which is two in this case.

```
absMoveSelectedCellsToBin "IO"
=> 2
```

Related Topics

Bin Preferences for Abstract Generation

absRenameBin

```
absRenameBin(
     name
     new_name
     )
     => t / nil
```

Description

Renames a previously created user bin.

Note: You cannot rename a system bin.

Arguments

name	Specifies the name of the user bin to be renamed.
new_name	Specifies the new bin name. Do not enter any spaces in the name.

Value Returned

0	Indicates an error.
1	Indicates successful operation.

Example

Renames the bin `userbin1` to `userbin2`.

```
absRenameBin "userbin1" "userbin2"
=> t
```

Related Topics

Bin Preferences for Abstract Generation

absNewBin

```
absNewBin(
     name
     { Core | IO | Corner | Block | Ignore }
     )
     => t / nil
```

Description

Creates a new user bin to store library cells.

Arguments

name	Specifies the name of the new user bin to create. Do not enter spaces in the name.
`{ Core | IO | Corner | Block | Ignore }`	Specifies the user bin type.

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error.

Example

Create a new user bin called `userbin1` which has the bin type `Core`.

```
absNewBin "userbin1" "Core"
=> t
```

Related Topics

Adding a User Bin in Abstract Generator

absDeleteBin

```
absDeleteBin(
     name
     )
     => t / nil
```

Description

Deletes the named user bin. You cannot delete a system bin.

Arguments

name	Specifies the name of a user bin to be deleted.

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error.

Example

Deletes the bin `userbin1`.

```
absDeleteBin "userbin1"
=> t
```

Related Topics

Deleting a User Bin in Abstract Generator

absDeleteBinMoveCellsTo

```
absDeleteBinMoveCellsTo(
     name
     { Core | IO | Corner | Block | Ignore | user_bin_name }
     )
     => t / nil
```

Description

Deletes a named user bin and moves any cells in that bin to another specified bin.

Arguments

name	Specifies the name of the user bin to be deleted.
{ Core | IO | Corner | Block | Ignore | *user_bin_name* }	Specifies the name of the bin into which any cells in the deleted bin are moved.

Value Returned

t	Indicates successful operation.
nil	Indicates an error.

Example

Deletes a user bin called `userbin1` and moves any cells previously contained in it to the `Core` bin.

```
absDeleteBinMoveCellsTo "userbin1" "Core"
```

Related Topics

Deleting a User Bin in Abstract Generator

absCopyBinOptions

```
absCopyBinOptions(
     from_bin { Core | IO | Corner | Block | Ignore | user_bin_name }
     to_bin { Core | IO | Corner | Block | Ignore | user_bin_name }
     )
     => t / nil
```

Description

Copies bin option settings from one bin to another.

Arguments

from_bin	Specifies the name of the bin from which to copy the option settings. Valid values include `Core`, `IO`, `Corner`, `Block`, `Ignore`, and *`user_bin_name`*.
to_bin	Specifies the name of the bin to which the option settings are to be copied.

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error. I

Example

Copies the option settings from the `Core` bin to the user bin called `userbin1`.

```
absCopyBinOptions "Core" "userbin1"
=> t
```

Related Topics

Copy Bin Options Form

absGetOption

```
absGetOption(
     name
     )
     => string
```

Description

Returns the value of the specified global option.

Arguments

name	Specifies the global option value to display. You can select from the following list: ■ Import Options ■ Export Options ■ Library Options ■ Select Options ■ Deselect Options ■ Distribute Options ■ Sort Options

Value Returned

string	Returns the value of the specified option.

Options

`ExtractShapeLimit`

Returns the limit set for the number of shapes to be extracted in the given design. The default value is `10000`. To increase the limit, use this option with the `absSetOption` function.

`ExcludePurposeList` *`purpose_names`*

Returns the names of purposes that are to be ignored when a layer name without a qualifying purpose name is specified in the geometry specifications in the Pin, Extract, and Abstract steps.

`ExtractPurposeList` *`purpose_names`*

Returns the names of the purposes that are extracted when a layer name without a qualifying purpose name is specified in the geometry specifications in the Extract step during signal extraction, power extraction, and extraction for antenna calculation.

Example:

```
absGetOption( "ExtractPurposeList"
"drawing net pin")
```

`UseConstraintGroup` *`constraintgroup_name`*

Returns the name of the technology database constraint group that Abstract Generator uses to get information such as valid vias and valid routing layers.

Examples

```
absGetOption "DefaultBin"
=> Core
```

Returns the current value set for the `DefaultBin` option.

Related Topics

Specifying Global Options in General Options form

absSetOption

```
absSetOption(
     name
     value
     )
     => 0 / 1
```

Description

Sets values for all abstract generator global options. The SKILL function to apply the `absSetOption` settings is run after the `absSetOption` function has been set.

Arguments

name	Specifies the option to be set. You can select from the following list: ■ Import Options ■ Export Options ■ Library Options ■ Select Options ■ Deselect Options ■ Distribute Options ■ Sort Options
value	Specifies the option value to be used. For the list of valid values for each of the above listed options, refer to the respective option details.

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Options

`ExtractShapeLimit` *`value`*

Sets the limit for the number of shapes to be extracted in the given design. The default value is `10000`. To increase the limit, set the value as a positive number greater than `10000`.

`ExcludePurposeList` *`purpose_names`*

Returns the names of purposes that are to be ignored when a layer name without a qualifying purpose name is specified in the geometry specifications in the Pin, Extract, and Abstract steps.

`ExtractPurposeList` *`purpose_names`*

Specifies the names of the purposes that are to be extracted when a layer name without a qualifying purpose name is specified while running Abstract Generator. This option is available as a global option and is applicable to signal extraction, power extraction, and antenna calculation.

`UseConstraintGroup` *`constraintgroup_name`*

Specifies the name of the technology database constraint group that Abstract Generator uses to get information such as valid vias and valid routing layers.

`OptimizeBigDesignMemory`

Solves the memory issue which comes on the big block designs in Abstract Generator. By default, this option is set to False. To optimize the memory for big designs, set this option to True.

Note: Even when the `OptimizeBigDesignMemory` option is set to true and you still have memory issue with large design size on 32bit executable, suggest running the 64 bit executable.

`QuickAbstract`

Improves performance of Abstract Generator for small and mid size designs. If this option is set to true, Abstract Generator will not generate the intermediate cellviews, that is Pin and Extract cellviews. Only Abstract cellviews will be generated. This will result in 30% performance gain. By default, this option is set to false.

`AnnotateLayoutDualView`

Enables Abstract Generator to annotate the abstract data in the layout itself and generate the layout dual view instead of the regular abstract view if set to `TRUE`. By default, this option is set to `FALSE`.

`ReadVoltageAndPurposeRules`

Enables Abstract Generator to read the layer-purpose-based and voltage dependent spacing rules from the technology file if set to `TRUE`. By default, this option is set to `FALSE`.

```
absSetOption(
"ReadVoltageAndPurposeRules" "true")
```

`CopyMSRoutingConstraints`

Enables Abstract Generator to copy the mixed signal routing constraints from the layout view to the abstract view, if set to `TRUE`. By default, this option is set to `FALSE`.

```
absSetOption("CopyMSRoutingConstraints"
"true")
```

`ReadOnlyTechnology`

Enables Abstract Generator to work with the read-only technology file. When this option is set to `TRUE`, Abstract Generator does not try to save to the technology file the layer purpose pairs that it creates. By default, this option is set to `FALSE`.

```
absSetOption("ReadOnlyTechnology" "true")
```

`numOfPointsInEllipseOrDonut`

Specifies the number of points that polygons should have before fracturing the blockages. These polygons represent elliptical, circular, or donut shapes in the layout. By default, this option is set to `64`.

```
absSetOption("numOfPointsInEllipseOrDonut
" "64")
```

`LEFLibraryName`

Specifies the LEF library that needs to be imported and displayed in the current abstract session.The specified LEF library is opened in Abstract Generator after reading the corresponding technology LEF file. At this point, the corresponding GDS file, if available, will also be read.

```
absSetOption("LEFLibraryName" "t8")
```

`suppressMessage`

Suppresses the messages that have the specified message IDs.

```
absSetOption("suppressMessage" "ABS-
12005" "ABS-15009")
```

`IgnoreCellsForExtraction`

Specifies the cells to be ignored during extraction. The valid format for the cell names is a space separated string.

```
absSetOption("IgnoreCellsForExtraction"
"CELL_55 CELL_56")
```

`ExtractPurposeList` *`purpose_names`*

Specifies the names of the purposes that are to be extracted when a layer name without a qualifying purpose name is specified in the geometry specifications in the Extract step during signal extraction, power extraction, and extraction for antenna calculation.

Example:

```
absSetOption( "ExtractPurposeList"
    "drawing net pin")
```

Example

Sets the `Block` bin to be the default bin.

```
absSetOption "DefaultBin" "Block"
=> 1
```

Related Topics

Specifying Global Options in General Options form

absGetBinOption

```
absGetBinOption(
     bin
     option
     )
     => string
```

Description

Returns the value of a specified option in a given bin.

Arguments

bin	Specifies the name of the bin with which the option is associated. Valid values include `Core`, `IO`, `Corner`, `Block`, `Ignore`, and *`user_bin_name`*.
option	Specifies the option value to display. You can select from the following list: ■ Pin Options ■ Extract Options ■ Abstract Options ■ Verify Options

Value Returned

string	Returns the value of the specified option.

Examples

Displays the name of the power pins in the `Core` bin.

```
absGetBinOption "Core" "PinsPowerNames"
=> ^((V(DD|CC))|(v(dd|cc)))(!)?$
```

Related Topics

Specifying Global Options in General Options form

absSetBinOption

```
absSetBinOption(
     bin
     option
     value
     )
     => t / nil
```

Description

Sets a value for a specified option in a specified bin.

Arguments

bin	Specifies the name of the bin for which the option is to be set. Valid values include `Core`, `IO`, `Corner`, `Block`, and `Ignore`.
option	Specifies the name of the option to set. You can select from the following list: ■ Pin Options ■ Extract Options ■ Abstract Options ■ Verify Options
value	Specifies the value for the specified option. For the list of valid values for each of the above listed options, refer to the respective option details.

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error.

Examples

Sets the abstract option `AbstractAdjustBoundaryPinsPwr` on.

```
absSetBinOption "Core" "AbstractAdjustBoundaryPinsPwr" "true"
=> t
```

Related Topics

Specifying Global Options in General Options form

absPins

```
absPins(
     )
     => 0 / 1
```

Description

Runs the Pins step for the selected cells and creates a pins view for those cells based on options set using `absSetBinOption`.

Arguments

None

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Options

`AbstractRemoveRedundantBlockages { true | false }`

Controls whether Abstract Generator removes redundant blockages, where the shapes of the same layer coincide with each other.

- If `true`, Abstract Generator removes redundant blockages. If `PinsPreserveRoutingBlockages` is also `true`, local blockages in the layout of the cell are preserved.
- If `false`, Abstract Generator does not remove redundant blockages.

`PinsTextPinMap` *`text_pin_spec`*

Maps text labels in the layout view to pins on nets in the pins view.

`PinsPowerNames` *`power_pin_names`*

Identifies power pin names based on label strings in the abstract. Enter a list of possible names for the power pins. This can be a list of regular expressions, each separated by a space.

`PinsGroundNames` *`ground_pin_names`*

Identifies ground pin names based on label strings in the abstract. Enter a list of possible names for the ground pins. This can be a list of regular expressions, each separated by a space.

`PinsClockNames` *`clock_pin_names`*

Identifies clock pin names based on label strings in the abstract. Enter a list of possible names for the clock pins. This can be a list of regular expressions, each separated by a space.

`PinsAnalogNames` *`analog_pin_names`*

Identifies analog pin names based on label strings in the abstract. Enter a list of possible names for the analog pins. This can be a list of regular expressions, each separated by a space.

`PinsOutputNames` *`output_pin_names`*

Identifies output pin names based on label strings in the abstract. Enter a list of possible names for the output pins. This can be a list of regular expressions, each separated by a space.

`PinsTextPromoteLevel` *`integer`*

Controls how far down the design hierarchy Abstract Generator searches for text labels during text-to-pin mapping.

`PinsGeomSearchLevel` *`integer`*

Controls how far down the design hierarchy Abstract Generator searches for metal underneath a text label.

`PinsTextManipulation` *`text_to_remove text_to_replace`*

Uses regular expressions to locate unnecessary text in labels and either remove it completely or replace it with the text you require.

Every odd regular expression represents *text_to_remove*, while every even regular expression represents *text_to_replace*.

To remove text completely, use {}as the *text_to_replace* element.

`PinsTextPreserveLabels { true | false }`

Preserves text labels from the layout view in the final abstract.

`PinsBoundaryCreate { off | always | as needed }`

Specifies whether Abstract Generator is to create a place-and-route boundary based on the layers supplied for the `PinsBoundaryLayers` argument.

- `off` means that Abstract Generator never creates a place-and-route boundary and reports an error if the boundary is missing from the layout.
- `always` means that Abstract Generator always creates a place-and-route boundary, after deleting any already existing boundary. It checks for the existence of a boundary on LPP `prboundary boundary`, deletes if one is found, and then creates the new boundary. If none exists on LPP `prboundary boundary`, it then searches for an existing boundary on LPP `prboundary drawing`, and deletes if any is found before creating the new boundary.
- `as needed` means that Abstract Generator creates a place-and-route boundary only if one does not already exist on LPP `prboundary boundary`.

The new place-and-route boundary created is rectangular. To allow for the creation of a rectilinear place-and-route boundary, use the `PinsCreatePolyPRB` option.

`PinsBoundaryLayers` *layers*

Specifies the layers to be used in calculating the place-and-route boundary. The boundary is drawn so that it encloses all the geometry found on these layers.

The argument is valid only if you set `PinsBoundaryCreate` to `always` or `as needed`.

`PinsCreatePolyPRB { true | false }`

Specifies the creation of a rectilinear place-and-route boundary. By default, it is set to `false`, in which case a rectangular place-and-route boundary is created.

The argument is valid only if you set `PinsBoundaryCreate` to `always` or `as needed`.

`PinsBoundarySizeLeft` *`value`*

Specifies the distance in microns to be added to the left boundary edge. This is one of four boundary options to let you increase or decrease the size of an existing or calculated place and route boundary.

Entering a positive number for this option moves the boundary further to the left by that amount; i.e., the lower left coordinate of the boundary is decreased.

This option is exclusive with the `PinsBoundaryFixedLeft` option.

`PinsBoundarySizeRight` *`value`*

Specifies the distance in microns to be added to the right boundary edge. This is one of four boundary options to let you increase or decrease the size of an existing or calculated place and route boundary.

This option is exclusive with the `PinsBoundaryFixedRight` option.

`PinsBoundarySizeTop` *`value`*

Specifies the distance in microns to be added to the top boundary edge. This is one of four boundary options to let you increase or decrease the size of an existing or calculated place and route boundary.

This option is exclusive with the `PinsBoundaryFixedTop` option.

`PinsBoundarySizeBottom` *`value`*

Specifies the distance in microns to be added to the bottom boundary edge. This is one of four boundary options to let you increase or decrease the size of an existing or calculated place and route boundary.

This option is exclusive with the `PinsBoundaryFixedBottom` option.

`PinsBoundaryFixedLeft` *`value`*

Fixes the left boundary edge of a cell to the specified value.

This option is exclusive with the `PinsBoundarySizeLeft` option.

`PinsBoundaryFixedRight` *`value`*

Fixes the right boundary edge of a cell to the specified value.

This option is exclusive with the `PinsBoundarySizeRight` option.

`PinsBoundaryFixedTop` *`value`*

Fixes the top boundary edge of a cell to the specified value.

This option is exclusive with the `PinsBoundarySizeTop` option.

`PinsBoundaryFixedBottom` *`value`*

Fixes the bottom boundary edge of a cell to the specified value.

This option is exclusive with the `PinsBoundarySizeBottom` option.

`PinsPreserveRoutingBlockages { true | false }`

Controls how Abstract Generator processes local blockages defined in the `BLOCKAGE` section in DEF 5.4. Local blockages do not have the `+PUSHDOWN`, `+FILL`, or `+SLOT` modifiers.

- If `false`, Abstract Generator removes any local blockages and proceeds as if they had never been specified in the DEF file.
- If `true`, and if `AbstractRemoveRedundantBlockages` is also `true`, Abstract Generator preserves local blockages along with their attributes from level 0 to level 32. The preserved layout blockages exists in the abstract view along with the other blockages generated in the abstract step.

Note: To preserve the routing blockages of layout, you need to delete the `boundary` purpose from the `ExcludePurposeList` option. For this, set the absSetOption as `absSetOption (ExcludePurposeList " ")`.

`PinsCreatePwrPinsFromRouting { true | false }`

Controls whether pins are created from the routing defined in the `SPECIALNETS` section of the DEF file.

- If `false`, Abstract Generator creates pins only from the pin geometry defined in the `PINS` section of the DEF file. If there is no pin geometry, the routing is turned into pins instead.
- If `true`, Abstract Generator creates pins from the routing defined in the `SPECIALNETS` section for the layers specified for the `PinsPwrRoutingLayers` option. It does this even if there are pins already defined in the `PINS` section.

When creating pins, Abstract Generator ignores `FILLWIRE` and `SHIELD` net segments. It also creates pins in instances that are logically connected to the top-level nets.

`PinsPwrRoutingLayers` *`layers`*

Specifies the layers to be used in the creation of pins from routing defined in the `SPECIALNETS` section of the DEF file.

`PinsRestrictToPRBndry { true | false }`

Controls the geometry of N-Well pins created during the abstract generation process. If set to t, the size of N-Well pins is restricted to the PRBoundary.

`PinsFromTextForExistingPins { true | false }`

When set to `true`, Abstract Generator generates additional pins from labels, even if the corresponding net has existing pins in the layout. The default value is false.

```
absSetBinOption( "Block"
"PinsFromTextForExistingPins" "true")
```

Examples

Specifies a text-to-pin map, for the `Core` bin, where labels on layer `Metal1` (all purposes) are to be mapped first to shapes on layer `Metal1` (purpose `drawing` only). If no shapes are found on `Metal1 drawing`, then the labels are to be mapped to shapes on layer `Metal2` (purpose `drawing` only). Finally, runs the Pins step.

```
absSetBinOption "Core" "PinsTextPinMap" "(Metal1 (Metal1 drawing)) (Metal2 (Metal2
drawing))"
=> t
absPins
=> 1
```

Related Topics

Specifying Pin Mappings for Abstract Generation

absExtract

```
absExtract(
     )
     => 0 / 1
```

Description

Runs connectivity extraction for the selected cells and creates an extracted view for these cells based on the options set using `absSetBinOption`.

Arguments

None

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Options

`ExtractSig { true | false }`

Specifies that connectivity information be extracted for signal nets, when set to `true`.

If `ExtractSig` is set to `false`, all other signal arguments are invalid except the `MustJoin` arguments.

`ExtractAntennaPinPathsOnly { true | false }`

Specifies that only paths that contain pin shapes from metal to gate or diffusion be considered for antenna calculation.

If set to `false` or if `ExtractSig` is set to `true`, all the shapes and paths are considered for antenna calculation whether they have pins or not.

`AbstractExtractGSpecTable` *`geom_spec`*

Specifies a geometric operation to establish the geometry or shapes that should be present on a particular layer when extraction is performed.

See the Layer Assignment for Signal Extraction table.

`ExtractLayersSig` *`layers`*

Specifies the layers through which the extractor extracts each net. To specify this argument, first set the `ExtractSig` option to `true`.

`ExtractLayersSigWeak { Strong | Weak }`

Selects the connectivity setting for a particular layer.

This argument is valid only if you have provided the associated layer and geometry specification.

`ExtractPinLayersSig` *`layers`*

Determines which of the shapes found by the extractor should be turned into pins.

This argument is valid only if the associated layer, geometry specification and connectivity have been provided.

`ExtractNumLevelsSig` *`max_depth`*

Controls how far down the design hierarchy the extractor searches for shapes. The default is `32`, the maximum depth of hierarchy in the database.

`ExtractDistSig` *`distance`*

Restricts the shapes considered by the extractor to those that intersect a bounding box around each starting pin shape. Specify the size of the bounding box in microns.

`ExtractWidthSig` *`width`*

Specifies the minimum width of shape to extract. The extractor extracts only shapes that are greater than or equal to the width you supply.

`ExtractMustConnectAllPinsAlways { true | false }`

Establishes a must-connect relation between all disjoint groups of shapes extracted for any net in the layout.

`ExtractMustConnectTerminals` *`terminals`*

Specifies a list of terminal names, each separated by a single space or a comma.

Establishes a must-connect relation between disjoint groups of shapes extracted for the terminals listed. For all pins in the net, Abstract Generator retains the same terminal name.

To specify this argument, set `ExtractMustConnectAllPinsAlways` to `false`.

`ExtractMustJoinAlways { true | false }`

Establishes a must-join relationship between all disjoint groups of geometries extracted from separate starting pins on the same net.

`ExtractMustJoinTerminals` *`terminals`*

Specifies a list of terminal names, each separated by a single space or a comma.

If the extractor finds a terminal name matching one of the names supplied, and if there are disjoint geometries from different starting pins on that net after extraction, then Abstract Generator creates a `MUSTJOIN` relationship for that terminal. To specify this argument, set the `ExtractMustJoinAlways` to `false`.

`ExtractDisableShortDetection { true | false }`

Specifies whether or not short detection is run by the extractor during the signal or power extraction process.

Set `ExtractDisableShortDetection` to `true` to run the extraction process without short detection.

`ExtractPwr { true | false }`

Specifies whether or not to extract connectivity for power nets.

`AbstractExtractPwrGSpecTable` *`geom_spec`*

Specifies a geometric operation to establish the geometry and shapes that should be present on a particular layer when extraction is performed.

For information about creating a geometry specification, see the Layer Assignment for Power Extraction table.

`ExtractLayersPwr` *`layers`*

Specifies the layers through which the extractor extracts each net. To specify this argument, first set the `ExtractPwr` option to `true`.

`ExtractPinLayersPwr` *`layers`*

Determines which of the shapes found by the extractor should be turned into pins.

This argument is valid only if the associated layer and geometry specification have been provided.

`ExtractNumLevelsPwr` *`max_depth`*

Controls how far down the design hierarchy the extractor searches for shapes. The default is `32`, the maximum depth of hierarchy in the database.

`ExtractDistPwr` *`distance`*

Restricts the shapes considered by the extractor to those that intersect a bounding box around each starting pin shape. Specify the size of the bounding box in microns.

`ExtractWidthPwr` *`width`*

Specifies the minimum width of shape to extract. The extractor extracts only shapes that are greater than or equal to the width you supply.

`ExtractAntennaHier { true | false }`

Calculates antenna hierarchically by summing the antenna data on standard abstracts.

`ExtractAntennaSizeInput { true | false }`

Calculates either the ANTENNASIZE LEF (if you are using LEF 5.3) or the ANTENNAGATEAREA attribute (if you are using LEF 5.4) for input pins.

`ExtractAntennaSizeOutput { true | false }`

Calculates either the ANTENNASIZE LEF (if you are using LEF 5.3) or the ANTENNAGATEAREA and ANTENNADIFFAREA attributes (if you are using LEF 5.4) for output pins.

`ExtractAntennaSizeInout { true | false }`

Calculates either the ANTENNASIZE LEF (if you are using LEF 5.3) or the ANTENNAGATEAREA and ANTENNADIFFAREA attributes (if you are using LEF 5.4) for IO pins.

`ExtractAntennaMetalArea { true | false }`

Calculates the area of metal that will be connected to the pin when the layer is fabricated.

`ExtractAntennaMetalSideArea { true | false }`

Calculates the side area of the metal connected to the pins is calculated (perimeter x thickness).

`ExtractAntennaMaxCutCARLowestCut { true | false }`

Includes CUT01 layer for the calculation of ANTENNAMAXCUTCAR.

`AbstractAntennaGSpecTable` *`geom_spec`*

Specifies a geometric operation to define the gate or drain geometry regions. For information about creating a geometry specification, see the Layer Assignment for Antenna Regions table.

`ExtractAntennaGate` *`geom_spec`*

Specifies a geometric operation to find the gate geometry; for example,

```
(Poly (Poly and Oxide))
```

`ExtractAntennaDrain` *`geom_spec`*

Specifies a geometric operation to determine the drain area that must be removed from the area of the diffusion; for example,

```
(Oxide (Oxide andnot Poly))
```

`ExtractAntennaExcludeCumPerLayerAntennaRatio { true | false }`

Extracts antenna factors by using the `antenna` constraint when set to `true`, its default value, or not set. When set to `false`, antenna factors are extracted using `cumPerLayerAntennaRatio`.

`ExtractAntennaOxide { Oxide1 | … | Oxide32 }`

Selects an oxide model from the available oxide types, which are in the range `Oxide1` through `Oxide32`. For example:

```
("Core" "ExtractAntennaOxide" "(OD Oxide12)")
```

`ExtractDiffAntennaLayers { true | false }`

Sets different layer assignments for antenna extraction.

`ExtractAntennaLayers` *`layers`*

Specifies the layers for which you want to extract antenna data.

This option is valid only if you are using LEF 5.4 and the `ExtractDiffAntennaLayers` option is set to `true`.

For information about creating a geometry specification, see the *Layer Assignment for Antenna Extraction* table.

`ExtractAntennaGSpecTable` *`geom_spec`*

Specifies a geometric operation to define antenna regions.

For information about creating a geometry specification, see the Layer Assignment for Antenna Regions table.

`ExtractConnectivity` *`layers`*

Specifies layers for which connectivity is to be extracted.

`ExtractAntennaIncludePolyCAR { true | false }`

Includes poly shape area in the calculation of `ANTENNAPARTIALMETALAREA`.

`ExtractAntennaIsSOIProcess { true | false }`

Specifies whether Fully Depleted Silicon-On-Insulator (FDSOI) pseudo gate area is to be calculated while calculating antenna data.

Example

Specifies that power nets should be extracted for `Metal1`, `Metal2`, and `Metal3` layers for cells in the core bin and then runs the Extract step.

```
absSetBinOption "Core" "ExtractPwr" "true"
=> t
absSetBinOption "Core" "ExtractLayersPwr" "Metal1 Metal2 Metal3"
=> t
absExtract
=> 1
```

Related Topics

Extract Step in Standalone Abstract Generation

absAbstract

```
absAbstract(
     )
     => 0 / 1
```

Description

Runs the Abstract step for selected cells and creates an abstract view for those cells based on the options set using `absSetBinOption`.

Arguments

None

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Options

`AbstractAdjustBoundaryPinsSig { true | false }`

Makes the signal pin shapes created during extraction square.

`AbstractAdjustBoundaryPinsPwr { true | false }`

Makes the power pin shapes created during extraction square.

`AbstractAdjustBoundaryPinsSigDist` *`distance`*

Specifies the distance (in microns) at which the signal net geometries should be from the boundary so that they can be considered as boundary pins. The geometries that are within the specified distance from the boundary are squared.

By default, the shape(s) closest to the boundary is squared.

`AbstractAdjustBoundaryPinsPwrDist` *`distance`*

Specifies the distance (in microns) at which the power net geometries should be from the boundary so that they can be considered as boundary pins. The geometries that are within the specified distance from the boundary are squared.

By default, the shape(s) closest to the boundary is squared.

`AbstractAdjustCreateClassBumpPort { true | false }`

Specifies if CLASS BUMP ports need to be created for a library.

If the option is set to `true`, bump ports are created for the pins whose names are mentioned in the `AbstractAdjustClassCoreNets` option. If the option is set to `false`, no bump ports are created.

`AbstractAdjustRingPinsPwr { true | false }`

Creates ring pins where any shapes in a power net are found to form a ring.

This argument is valid only when processing blocks and is mutually exclusive with the `AbstractAdjustBoundaryPinsPwr` argument.

`AbstractAdjustRingPinsDist` *`distance`*

Specifies in microns the maximum distance from the boundary that extracted power net geometry can be in order for it to be considered part of the ring and therefore to be included in the abstract as pin geometry.

This argument is valid only if you are processing a block and if `AbstractAdjustRingPinsPwr` is `true`.

`AbstractAdjustFollowRingPin { true | false }`

Specifies that all pin shapes, including the ones that do not form the ring, are considered as pins in the final abstract.

If the option is set to `true` and no value is specified for the `AbstractAdjustRingPinsDist` option, only the pin shapes whose distance from the boundary is the same or less than the distance of the farthest edge of the ring pin shape are included in the abstract. If the option is set to `true` and a value is specified for the `AbstractAdjustRingPinsDist` option, pin shapes are included in the abstract only if their distance from the boundary is within the distance specified for the option.

Specify an arbitrarily large value for the `AbstractAdjustRingPinsDist` option so that no filtering happens based on the distance you specify.

```
AbstractAdjustPowerGeometryGroups { single | separate | overlap }
```

Controls how power shapes are grouped into LEF PORTS.

- `single` creates a single LEF PORT for all the power shapes.
- `separate` creates a separate LEF PORT for each power shape.
- `overlap` forms LEF PORTS from groups of overlapping power pins. This means that the power shapes that overlap go into the same LEF PORT.

```
AbstractAdjustPowerGeometryGroupsEastWest { single | separate |
overlap | none }
```

Controls how power shapes touching the east or west edge of a cellview are grouped into LEF PORTS. The remaining shapes that do not touch the east or west side are grouped according to the *AbstractAdjustPowerGeometryGroups* option.

- `single` creates a single LEF PORT for all the power shapes touching the east or west edge.
- `separate` creates a separate LEF PORT for each power shape touching the east or west edge.
- `overlap` forms LEF PORTS from groups of overlapping power pins touching the east or west edge. This means that the power shapes that overlap go into the same LEF PORT.

By default, this option is set to `None`

A power shape touching the east or west edge as well as one of either the north or the south edge is grouped as per the *AbstractAdjustPowerGeometryGroupsEastWest* option. Even a shape touching the north and south edges in addition to touching either the east or the west edge is grouped as per the `AbstractAdjustPowerGeometryGroupsEastWest` option.

`AbstractAdjustEdgeTowardsCore`

Specifies the direction of the CORE-facing edge. This argument is available only when a pad cell in the IO bin is being processed. You can specify multiple directions of the CORE-facing edge. The valid values for the `CORE`-facing edge are, north, south, east, west, or a combination of any of the four core edges.

`AbstractAdjustClassCoreNets` *`net_name`*

Specifies which nets should have `CLASS CORE` ports. Accepts a regular expression; for example,

```
^((V(DD|CC))|(v(dd|cc)))(!)?$
```

`AbstractAdjustIncludeAllShapes { true | false }`

Specifies that multiple ports that are equidistant from the core-facing edge of a pad are considered as `CLASS CORE` ports. Only the geometries with their center lines perpendicular to the core-facing edge are considered. This option is applicable to pad cells that are processed in the IO bin.

This option is set to `false` by default, which means that only the geometry closest to the core-facing edge becomes a `CLASS CORE` port. When you set the option to `true`, all the geometries equidistant from the core-facing edge are considered as `CLASS CORE` ports.

`AbstractAdjustCopyClassCorePort { true | false }`

Duplicates any geometry found to belong to a `CLASS CORE` port on the net (set using the `AbstractAdjustClassCoreNets` argument) in a non-`CLASS CORE` port in the exported abstract LEF.

`AbstractAdjustPinsTouchBoundary { true | false }`

Specifies that only pins that touch the boundary of the cell become CLASS CORE ports in LEF.

`AbstractAdjustPreserveViasPwr { true | false }`

Preserves existing layouts rather than removing them when the `absAbstract` function is run.

`AbstractAdjustPowerRailOp` *`net_name`*

Specifies the net name whose power and ground rail characteristics you want to set.

`AbstractAdjustPowerRailOpTable` *`net_spec`*

Specifies the characteristics of the power and ground rails for the current bin.

The *net_spec* takes the form

```
netName shape width offset
```

- *netName* is the name of a net.
- *shape* can be one of `calculate`, `feedthru`, `abutment` or `None`
- *width* is the rail width in microns (must be non-negative float)
- *offset* is the rail offset in microns (any float)

`AbstractAdjustPowerRailWidth` *width*

Sets the rail width value for the net specified by `AbstractAdjustPowerRailOp`.

`AbstractAdjustTrimPins { true | false }`

Trims long pins that touch the PR boundary and extend deep inside the cell.

`AbstractAdjustTrimPinsFactor` *width*

Specifies the trim factor as an integral multiple of the width of the boundary pin.

`AbstractAdjustExcludeNetNames` *net_name*

Specifies which nets should be excluded.

Accepts a regular expression; for example,

```
^((V(DD|CC))|(v(dd|cc)))(!)?$
```

`AbstractAdjustAllowPin` *geom_spec*

Specifies a geometric operation to describe the regions where pins are to be allowed. If the `AllowAvoidSignalAndPowerPinsSeparately` Shell environment variable is set, this option is used only for signal pins. If not set, it is used for both signal and power pins.

`AbstractAdjustAvoidPin` *geom_spec*

Specifies a geometric operation to describe the regions where pins are to be avoided. If the `AllowAvoidSignalAndPowerPinsSeparately` Shell environment variable is set, this option is used only for signal pins. If not set, it is used for both signal and power pins.

`AbstractAdjustAllowPowerPin` *`geom_spec`*

Specifies a geometric operation to describe the regions where only power pins are to be allowed. This option works only if the `AllowAvoidSignalAndPowerPinsSeparately` Shell environment variable is set.

`AbstractAdjustAvoidPowerPin` *`geom_spec`*

Specifies a geometric operation to describe the regions where only power pins are to be avoided. This option works only if the `AllowAvoidSignalAndPowerPinsSeparately` Shell environment variable is set.

`AbstractBlockageTable` *`layer_names`*

Specifies the layers on which to create blockages. This controls the final blockage geometry in the abstract.

`AbstractBlockageDetailedLayers` *`layer_names`*

Specifies the layers for which a detailed blockage model is to be created. The detailed blockage model generates blockages only where there are real shapes in the cell on the layer.

`AbstractBlockageCoverLayers` *`layer_names`*

Specifies the layers for which a cover blockage model is to be created. A cover blockage blocks the entire area that is not occupied by pin shapes, effectively blocking a layer for routing

`AbstractBlockageShrinkWrapLayers` *`layer_names`*

Specifies the layers for which a shrink wrap blockage model is to be generated. The shrink wrap blockage model fills in smaller, less useful free spaces and leaves larger spaces in the cell open. This allows over-the-cell routing without modeling each obstruction individually.

`AbstractBlockageCutAroundPin`

Controls whether the blockage around a pin is cut for allowing the router to access the pin. If this option is enabled, the blockage is cut at `min-spacing` distance around pins.

`AbstractBlockageMaxRoutingSpace`

Controls whether blockage is to be created around a block to drive the external routes maximum distance away from the block.

This option is valid only if `AbstractBlockageCutAroundPin` has been enabled. When both `AbstractBlockageCutAroundPin` and `AbstractBlockageMaxSpacingRoute` are enabled, the `AbstractBlockageCorridorCut` option is ineffective even if enabled.

`AbstractBlockageCorridorCut`

Controls whether corridors are cut out around pins for allowing routers to access internal pins.

This option is valid only if `AbstractBlockageCutAroundPin` has been enabled. This option is ineffective when both `AbstractBlockageCutAroundPin` and `AbstractBlockageMaxSpacingRoute` have been enabled.

`AbstractBlockagePinCutWindow` *`distance`*

Specifies the distance to cut around pins on the same layer. If set as `0`, the blockage completely covers the design, including the pin shape, and there is no pin cut out.

`AbstractBlockageDownPinCutWindow` *`distance`*

Specifies the distance to cut around pins on the layer below.

`AbstractBlockageCutForAboveLayers`

Specifies a valid metal layer that contains pins and the size of the pin cutouts to be created in the blockage layers above this layer. For example,

```
absSetBinOption("Block"
"AbstractBlockageCutForAboveLayers" "Metal2
(Metal4 0.2)")
```

`AbstractBlockageRoutingChannelOnLayers`

Specifies the shape of the routing channel to be created when generating the cover blockage. For example,

```
absSetBinOption ("Block"
"AbstractBlockageRoutingChannelOnLayers"
"(Metal1 (y0 drawing)) (Metal2 (y0 drawing))")
```

`AbstractBlockageShrinkAdjust` *`distance`*

Specifies a minimum distance in microns between blockages. Blockages separated by a distance less than or equal to the specified distance are merged.

This option is valid only for layers specified by `AbstractBlockageShrinkWrapLayers`.

`AbstractBlockageShrinkTracks` *`number_of_tracks`*

Specifies the minimum number of tracks between blockages. Blockages separated by a distance less than or equal to the specified number of tracks are merged.

This option is valid only for layers specified by `AbstractBlockageShrinkWrapLayers`.

`AbstractBlockageUserDefinedDistanceForAddBack` *`distance`*

Specifies the distance within which the geometries need to be added back when Abstract Generator creates a cover blockage with pin cutout.

`AbstractBlockageUserDefinedWidth` *`width`*

Sets the effective width attribute for layer blockages. This value can be set only if the spacing value is set to zero. The `AbstractBlockageUserDefinedWidth` option takes only positive float values, for example:

```
absSetBinOption( "Block"
"AbstractBlockageUserDefinedWidth"
"(Metal1 5.0)  (Metal2 4.0)")
```

`AbstractBlockageCoverLayersDist` *`distance`*

Specifies the minimum distance between a cover blockage and the cell boundary. If you do not specify a distance, Abstract Generator uses a distance equal to one half of the minimum layer separation.

This option is valid only for layers specified by `AbstractBlockageCoverLayers`.

`AbstractZeroSpacingBlockage { true | false }`

By default, the option is set to `false`. The working of this option depends on the value set for the option `AbstractBlockagePinCutWindow`:

- `AbstractBlockagePinCutWindow` is non-zero or is not specified

  In this case, the following situations are possible:

  - `AbstractZeroSpacingBlockage` is `true`: The blockage around the pin is cut out at zero spacing from the pin.
  - `AbstractZeroSpacingBlockage` is `false`: The blockage around the pin is cut out at default spacing from the pin.

- `AbstractBlockagePinCutWindow` is `0`

  If set as `0`, this option overrides any value set for the option `AbstractZeroSpacingBlockage`. In this scenario, the blockage completely covers the design, including the pin shape, and there is no pin cut out.

`AbstractDensity { true | false }`

Specifies whether Abstract Generator is to generate the metal density information. The default value of this option is `false`.

`AbstractDensityUseSignalLayers { true | false }`

Directs Abstract Generator to use the same geometry specifications for metal density calculation as are specified for signal extraction using the `ExtractLayersSig` option.

The default value of `AbstractDensityUseSignalLayers` is `false`.

Setting this option to `true` automatically sets the `AbstractDensityUsePwrLayers` and `AbstractDensityUseAntennaLayers` options to `false`.

`AbstractDensityUseAntennaLayers { true | false }`

Directs Abstract Generator to use the same geometry specifications for metal density calculation as will be used for process antenna calculation. In other words, to specify the layers and their geometry specifications, use the `ExtractAntennaLayers` option if the `ExtractDiffAntennaLayers` option is set to `true`; otherwise use the `ExtractLayersSig` option.

The default value of `AbstractDensityUseAntennaLayers` is `false`.

Setting this option to `true` automatically sets the `AbstractDensityUseSignalLayers` and `AbstractDensityUsePwrLayers` options to `false`.

`AbstractDensityUsePwrLayers { true | false }`

Directs Abstract Generator to use the same geometry specifications for metal density calculation as are specified for power net extraction using the `ExtractLayersPwr` option.

The default value of `AbstractDensityUsePwrLayers` is `false`.

Setting this option to `true` automatically sets the `AbstractDensityUseSignalLayers` and `AbstractDensityUseAntennaLayers` options to `false`.

`AbstractDensityLayers` *`layer_name geom_spec`*

Specifies the exclusive metal layers along with their geometry specifications that are to be considered by Abstract Generator for metal density calculation.

For example,

```
AbstractDensityLayers "Metal1 Metal2
Metal3 (Metal4 (Metal4 drawing)) Metal5
Metal6"
```

The default value for this option is a list of all metal layers:

```
AbstractDensityLayers "Metal1 Metal2
Metal3 Metal4 Metal5 Metal6"
```

`AbstractDensityWindowWidth` *`layer_name value`*

Specifies the width of the density window for a particular layer. For example,

```
AbstractDensityWindowWidth "(Metal1
10.0)"
```

If you do not specify a value for a layer, the value specified for the `AbstractDensityDefaultWindowWidth` option is taken as the default.

`AbstractDensityWindowHeight` *`layer_name value`*

Specifies the height of the density window for a particular layer. For example,

```
AbstractDensityWindowHeight "(Metal1 10.0)"
```

If you do not specify a value for a layer, the value specified for the `AbstractDensityDefaultWindowHeight` option is taken as the default.

`AbstractDensityDefaultWindowWidth` *`value`*

Specifies the default width of the density window when no width value is specified for a particular layer. For example,

```
AbstractDensityDefaultWindowWidth "30.0"
```

You can override the default value (`20.0`) for the width of the density window by using this option.

`AbstractDensityDefaultWindowHeight` *`value`*

Specifies the default height of the density window when no height value is specified for a particular layer. For example,

```
AbstractDensityDefaultWindowHeight "40.0"
```

You can override the default value (`20.0`) for the height of the density window by using this option.

`AbstractPinFracture { true | false }`

Controls whether Abstract Generator creates each pin shape as a set of maximum rectangles or as a polygon. This lets you model 45-degree pins if your design requires it.

- If `true`, Abstract Generator creates each pin shape as a set of maximum rectangles. This is recommended if the abstracts are to be routed using Silicon Ensemble or Encounter.
- If `false`, Abstract Generator creates each blockage shape as a polygon. This is the default setting and is recommended if the abstracts are to be routed using the IC Shape-Based Router.

`AbstractBlockageFracture { true | false }`

Controls whether Abstract Generator creates each blockage shape as a set of maximum rectangles or as a polygon.

- If `true`, Abstract Generator creates each blockage shape as a set of maximum rectangles. This is recommended if the abstracts are to be routed using Silicon Ensemble or Encounter.
- If `false`, Abstract Generator creates each blockage shape as a polygon. This is the default and is recommended if the abstracts are to be routed using the IC Shape-Based Router.

`AbstractAdjustStairStepCover { full | truncate | partial }`

Controls the degree to which stair-step blockages cover 45-degree geometry.

`AbstractAdjustStairStepWidth` *`width`*

Specifies the width of the blockage geometry to be created to cover 45-degree shapes.

`AbstractSiteName` *`site_name`*

Specifies a site name to be applied to all cells in the current bin.

`AbstractSiteNameDefine` *`site_name`*

Specify the name of a new site to be used by all cells in the current bin. If the site does not already exist, Abstract Generator defines it in the technology file.

`AbstractSiteArrayPattern { true | false }`

Specifies if Abstract Generator is to calculate site pattern for the gate-array sub-cellviews specified in the option `AbstractSiteArrayCells`.

`AbstractSiteArrayCells` *`sub-cellview site`*

Specifies the mapping of sub-cellviews to sites for calculating site patterns.

`AbstractOverlapLayerAction { off | always | as needed }`

Specifies whether Abstract Generator is to create an overlap boundary.

- `always` means that Abstract Generator always creates an overlap boundary. Any existing overlap layer geometry is overwritten.
- `as needed` means that Abstract Generator creates an overlap boundary if it is not already present or if it is present and was previously calculated by Abstract Generator.
- `off` means that Abstract Generator never creates an overlap boundary.

`AbstractOverlapLayers` *`layers`*

Specifies the layers to be used in calculating the overlap layer boundary. The boundary is drawn so that it encloses all the geometry found on these layers.

The argument is valid only if you set `AbstractOverlapLayerAction` to `always` or `as needed`.

`AbstractOverlapLayerSize` *`size`*

Specifies in microns how much an overlap layer is to be grown.

Use this option to ensure that the placement of cells with abutting overlap layers does not lead to geometry contained in these cells causing violations.

`AbstractOverlapLayerSmoothFactor` *`size`*

Controls the shape of the overlap layer produced.

Use this option to create simpler overlap boundary shapes, typically when processing blocks. The abstract generator takes the computed overlap boundary and removes any cut-outs with a maximum dimension less than the specified distance.

`AbstractGridMode { report | calculate | calculate M1 | calculate M2 | off }`

Specifies the grid analysis mode for standard cell bins.

- In `report` mode, you can specify

  ```
  AbstractMetal1Pitch
  AbstractMetal1Offset
  AbstractMetal2Pitch
  AbstractMetal2Offset
  ```

  But not

  ```
  AbstractMetal1PitchPercent
  AbstractMetal2PitchPercent
  ```

- In `calculate` mode, you can specify

  ```
  AbstractMetal1PitchPercent
  AbstractMetal2PitchPercent
  ```

  But not

  ```
  AbstractMetal1Pitch
  AbstractMetal1Offset
  AbstractMetal2Pitch
  AbstractMetal2Offset
  ```

- In `calculate_M1` mode, you can specify

  ```
  AbstractMetal2Pitch
  AbstractMetal2Offset
  AbstractMetal1PitchPercent
  ```

  But not

  ```
  AbsabstractMetal1Pitch
  AbstractMetal1Offset
  AbstractMetal2PitchPercent
  ```

- In `calculate_M2` mode, you can specify

  ```
  AbstractMetal1Pitch
  AbstractMetal1Offset
  AbstractMetal2PitchPercent
  ```

  But not

  ```
  AbstractMetal2Pitch
  AbstractMetal2Offset
  AbstractMetal1PitchPercent
  ```

```
AbstractUpdateTechFile { true | false }
```

Controls whether the technology file is update with the calculated grid.

`AbstractMetal1Pitch` *`value`*

Specifies the `metal1` layer pitch value (for core cells only).

If you run `AbstractGridMode` in `report` mode, these are the values that are used in the report. If you run grid analysis in `calculate` mode, these values are overwritten with the new best grid values for the layers being calculated.

`AbstractMetal1Offset` *`value`*

Specifies the `metal1` layer offset value (for core cells only).

If you run `AbstractGridMode` in `report` mode, these are the values that are used in the report. If you run grid analysis in `calculate` mode, these values are overwritten with the new best grid values for the layers being calculated.

`AbstractMetal2Pitch` *`value`*

Specifies the `metal2` layer pitch value (for core cells only).

If you run `AbstractGridMode` in `report` mode, these are the values that are used in the report. If you run grid analysis in `calculate` mode, these values are overwritten with the new best grid values for the layers being calculated.

`AbstractMetal2Offset` *`value`*

Specifies the `metal2` layer offset value (for core cells only).

If you run `AbstractGridMode` in `report` mode, these are the values that are used in the report. If you run grid analysis in `calculate` mode, these values are overwritten with the new best grid values for the layers being calculated.

`AbstractMetal1PitchPercent` *`value`*

Sets the upper limit for pitches considered during pitch calculation for the `metal1` layer.

This argument is valid only if `AbstractGridMode` is using set to use one of the calculation values.

`AbstractMetal2PitchPercent` *`value`*

Sets the upper limit for pitches considered during pitch calculation for `metal2` layer.

This argument is valid only if `AbstractGridMode` is using set to use one of the calculation values.

`AbstractDiagonalVias { true | false }`

Controls whether or not the pitches calculated will let vias be placed diagonally adjacent to each other.

When this option is `true`, the calculated pitches are larger than they would otherwise be in order to accommodate diagonal via placement.

`AbstractGridDistanceMetric { maxxy | euclidean }`

Controls how the clearance check between a pin geometry and its neighboring geometries is applied.

- `euclidean` means that the distance between two rectangles is the shortest straight line between them.
- `maxxy` means that the distance is the greater of the vertical and horizontal distance between the two rectangles.

`AbstractPinStretchOutPrb { true | false }`

Allows pins to be stretched outside the PR boundary. By default, this option is off.

`AbstractBlockageEXCEPTPGNET`

This option is set for blocks that want to block routing on a given layer except for power and ground routing. This is typically true for analog or memory blocks that are sensitive to noise, and do not want to allow signal routing on the layers immediately above the block.

When the option is set, then all the blockages generated by Abstract on given metal layers will have the EXCEPTPGNET LEF attribute in the LEF file.

`AbstractTopLayerCoverBlockage {Routing Layer}`

Specifies the top metal layer for cover blockage in the layout dual view. This option is added only for Layout dual view and will not cause any conflict with the normal cover blockage option that Abstract Generator has. The already existing cover blockage options in Abstract Generator have different use model and may lead to confusion.

Note: The `AbstractTopLayerCoverBlockage` argument can only be used when the `AnnotateLayoutDualView` option is set to true.

`AbstractBlockageUserDefinedSpacing`

Allows you to specify SPACING on the blockages. The value of SPACING specified by this option will supersede the SPACING specified by Abstract Generator on the blockage. The default value for this option is `absSetBinOption("Core""AbstractBlockageUserDefinedSpacing" "")`. However, the expected value is `absSetBinOption("Core" "AbstractBlockageUserDefinedSpacing" "(M1 1.0) (M2 2.0)")`. This will specify the blockages of layer M1 with spacing 1.0 and blockages on layer M2 with spacing 2.0.

`AbstractFractureRectilinearShapes { true | false }`

Controls whether only rectilinear pin shapes are farctured.

`BlockageCutVia`

	When set to `true`, Abstract Generator cuts a window large enough for a via to be dropped to that pin. When set to `false`, Abstract Generator cuts a window around the pin based on the range-based spacing rule for the pin width.
`BlockageLargePurposeList`	
	Specifies the purposes of the big blockages.
`BlockageLargeShape`	
	Enables Abstract Generator to look for big blockages in the current design abstract.
`BlockageLargeShapePct`	
	Specifies the percentage by which the size of the big blockages needs to be increased.

Examples

Specifies a number of power net options for cells in the `Core` bin and runs the Abstract step.

```
absSetBinOption "Core" "AbstractAdjustBoundaryPinsPwr" "true"
=> t
absSetBinOption "Core" "AbstractAdjustPowerGeometryGroups" "single"
=> t
absAbstract
=> 1
```

Related Topics

Abstract Step in Standalone Abstract Generation

absVerify

```
absVerify(
     )
     => 0 / 1
```

Description

Runs the Verify step for selected cells and creates a verify view for those cells based on the options set using `absSetBinOption`.

Arguments

None

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Options

`VerifyTerminals { true | false }`

Compares the terminals found in the logical and abstract views and reports any differences.

`VerifyGrid { true | false }`

Identifies pins and geometries that are located both on and off the manufacturing grid for each cell.

`VerifyTarget { true | false }`

Generates a test design for selected cells and runs it through the place-and-route system specified using `VerifyTargetSelection`.

`VerifyTargetSelection { Silicon Ensemble | Encounter }`

Specifies the target place-and-route system.

`VerifyTargetSystemCMD` *`command_line_option`*

Specifies the command line used to launch the target system.

`VerifyTechFile` *`technology_file_name`*

Specifies a technology LEF file to be used by the target system.

`VerifyTargetMultiple { true | false }`

Places an additional instance of the cell being verified on either side of the main instance of the cell in the test design, provided the cell's symmetry allows it. This simulates same-side cell abutment and helps identify any pins or geometry that are too close to the cell boundary.

`VerifyTargetMultipleRows { true | false }`

Places cells above and below the main instance as follows.

- Flipped about the x-axis (for example, top to bottom), provided the cell's symmetry allows it
- Abutted, if the site definition for that cell has zero row spacing

`VerifyTargetPower { true | false }`

Creates a power ring around the instances in the test design and routes it in the target system.

Turning the option off means that only signal nets are routed.

`VerifyTargetRouter { Nanoroute | Wroute }`

Specifies the routing engine. The default routing engine is `Nanoroute`.

`VerifyTargetMaxRouteTime` *`value`*

Specifies in minutes the maximum amount of time that Wroute or Nanoroute spends routing the test design. If the design fails to route within a reasonable time, it might be an indication that some of pins in the abstract are difficult to access.

`VerifyConfigFile` *value*

Specifies the path of the configuration file for running route. Therefore, `VerifyTargetRouter` should be set to `Wroute` before using this option.

`VerifyTargetGeom` *value*

Specifies the `verifyGeometry` function options, which verify the geometry of the targeted abstracts. In case of geometry violations, this option creates markers corresponding to each of the violations in the Verify view. The default value is `verifyGeometry`.

`VerifyTargetIgnore` *message_identifiers*

Specifies a list of regular expressions which are used to identify target system messages to be ignored when creating warning and error markers in the verify view

Examples

Specifies that terminals found in the logical and abstract views are to be compared, and a test design is generated for all selected cells, and then runs the Verify step.

```
absSetBinOption "Core" "absVerifyTerminals" "true"
=> t
absSetBinOption "Core" "absVerifyTarget" "true"
=> t
absVerify
=> 1
```

Related Topics

Calculating Metal Layer Density

absGetCellProp

```
absGetCellProp(
     cell
     { prCellClass | symmetry }
     )
     => string
```

Description

Returns the specified property value for a cell.

Arguments

cell	Specifies the name of the cell you want to query.
{ prCellClass | symmetry }	Specifies the property whose value you want to know.

Value Returned

string	An error is encountered.

Examples

Displays the current `prCellClass` cell property value for `cellABC`.

```
absGetCellProp "cellABC" "prCellClass"
=> block
```

Related Topics

Select Cells form

absSetCellProp

```
absSetCellProp(
     cell
     { prCellClass | symmetry }
     value
     )
     => 0 / 1
```

Description

Specifies a property value for a cell. The variables you can use for the *value* parameter depend on the property.

Arguments

cell — Specifies the cell whose properties are to be set.

{ prCellClass | symmetry } — Specifies the cell property to be edited.

value — The valid values for the property prCellClass depend on the bin in which the cell is.

For the *Core* bin, choose from:

```
[ core | feedthru | tieHigh | tieLow | spacer |
antennaCell | preEndcap | postEndcap ]
```

For the *Block* bin, choose from:

```
[ block | cover | ring | coverBump | blackBox ]
```

For the *IO* bin, choose from:

```
[ pad | input | output | inout | power | spacer |
areaIO ]
```

For the *Corner* bin, choose from:

```
[ bottomLeftEndcap | bottomRightEndcap |
topLeftEndcap | topRightEndcap ]
```

The valid values for the property symmetry include:

```
[ R0 | X | Y | R90 | X Y | X R90 | Y R90 | X Y R90 ]
```

Value Returned

`0`	An error is encountered.
`1`	The function executes successfully.

Example

Sets the `prCellClass` property value for the Core cell `abc` to `core`.

```
absSetCellProp "abc" "prCellClass" "core"
=> 1
```

Related Topics

Select Cells form

absGetTerminalProp

```
absGetTerminalProp(
     cell
     terminal
     property { direction | termType | shape | netExpr | suppSensitivity |
     groundSensitivity }
     )
     => string
```

Description

Returns the value of the specified property for the specified terminal. Specify only cells that have already been processed by `absPins`. The Pins step generates the cell terminals.

Arguments

cell	Specifies the cell to query.
terminal	Specifies the terminal to query.
property { direction | termType | shape | netExpr | suppSensitivity | groundSensitivity }	Specifies the property whose value you want to know.

Value Returned

string	Returns the value of the specified terminal property.

Examples

Displays the current `direction` terminal property value for terminal `vdd!` in `cellABC`.

```
absGetTerminalProp "cellABC" "vdd!" "direction"
=> inout
```

Related Topics

Viewing and Editing Cell Terminal Properties in Abstract Generator

absSetTerminalProp

```
absSetTerminalProp(
     cell
     terminal
     property { direction | termType | shape | netExpr | suppSensitivity |
     groundSensitivity }
     value
     )
     => 0 / 1
```

Description

Specifies a terminal value for a cell. The variables you can use for the *value* parameter depend on the *property* setting.

Arguments

cell	Specifies the cell whose properties are to be set.
terminal	Specifies the terminal whose properties are to be set.
property { direction | termType | shape | netExpr | suppSensitivity | groundSensitivity }	Specifies the terminal property to be edited.
value	Specifies the value to be set for the terminal property specified through *property*. The valid values for different properties are given below. ■ direction: input, output, inout, jumper, tristate, unused ■ termType: signal, clock, analog, power, ground ■ shape: feedthru, abutment, ring ■ netExpr: In the format [@gnd:%:gnd!] ■ suppSensitivity: Specify a valid power terminal name. ■ groundSensitivity: Specify a valid ground terminal name.

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Examples

Sets the terminal `direction` property value for terminal `vd!` in `cellABC` to be `input`.

```
absSetTerminalProp "cellABC" "vdd!" "direction" "input"
=> 1
```

Related Topics

Viewing and Editing Cell Terminal Properties in Abstract Generator

absDisableUpdate

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

Description

Both the `absDisableUpdate` and `absEnableUpdate` functions are used to control whether or not the graphical user interface updates in response to changes in, for example, bin selection. This ensures that the GUI does not update unnecessarily when several events in a row are processed. This is inserted by Abstract Generator into the `record` file.

Arguments

None

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error.

Examples

Specifies that the GUI is not to be updated when the `Core` bin is deselected and the `Ignore` bin selected instead.

```
absDisableUpdate
=> t
absDeselectBinFrom "Core" "Core"
=> t
absSelectBinFrom "Ignore" "Ignore"
=> t
absEnableUpdate
=> t
```

Related Topics

Abstract Generator Log, Replay, and Record Files

absDistributeCells

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

Description

Distributes cells between bins according to the options set.

Arguments

None

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error.

Options

`DistributeBinBelowBottom` *`bin_name`*

Specifies the bin in which to place cells that are smaller than the height set using `DistributeHeightBottom`.

The default setting is `Ignore`.

`DistributeBinAboveBottom` *`bin_name`*

Specifies the bin in which to place cells that are bigger than the height set using `DistributeHeightBottom`.

The default setting is `Core`.

`DistributeBinAboveMiddle` *`bin_name`*

Specifies the bin in which to place cells that fall within the criteria set by the `DistributeHeightMiddle` option.

The default setting is `IO`.

`DistributeBinAboveTop` *`bin_name`*

Specifies the bin in which to place cells that fall within the criteria set using `DistributeHeightTop`.

The default setting is `Block`.

`DistributeHeightBottom` *`height`*

Specifies in microns the minimum height for cells to be distributed.

Cells that are larger than or equal to the specified height but less than the specified middle height are placed in the bin set by the `DistributeBinAboveBottom` option.

The default setting is `100`.

`DistributeHeightMiddle` *`height`*

Specifies in microns the middle height range for cells to be distributed.

Cells that are larger than or equal to the specified height but less than the specified top height are placed in the bin set by the `DistributeBinAboveMiddle` option.

The default setting is `180`.

`DistributeHeightTop` *`height`*

Specifies in microns the maximum height for cell distribution.

Cells larger than or equal to the specified height are placed in the bin set by the `DistributeBinAboveTop` option.

The default setting is `1000`.

`DistributeCells { all | selected }`

Specifies whether to distribute all cells or only those cells currently selected.

The default setting is `all`.

`DistributeMethod { name | height }`

Selects the method for distributing cells into bins. The default setting is `height`.

Examples

Sets the distribution method to be `height` for `all` cells in the current library. Any cells of a height equal to or above `100` microns are placed in `UserBin1`.

```
absSetOption "DistributeMethod" "height"
=> 1
absSetOption "DistributeCells" "all"
=> 1
absSetOption "DistributeHeightTop "1000"
=> 1
absSetOption "DistributeBinAboveTop "UserBin1"
=> 1
absDistributeCells
=> t
```

Related Topics

Distribute Cells Among Bins Form

absEnableUpdate

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

Description

Both the `absEnableUpdate` and `absDisableUpdate` functions are used to control whether or not the graphical user interface updates in response to changes in, for example, bin selection. This ensures that the GUI does not update unnecessarily when several events in a row are processed. This is inserted by Abstract Generator into the `record` file.

Arguments

None

Value Returned

`t`	Indicates successful operation.
`nil`	Indicates an error.

Examples

Specifies that the GUI is not to be updated when the `Core` bin is deselected and the `Ignore` bin selected instead.

```
absDisableUpdate
=> t
absDeselectBinFrom "Core" "Core"
=> t
absSelectBinFrom "Ignore" "Ignore"
=> t
absEnableUpdate
=> t
```

Related Topics

Abstract Generator Log, Replay, and Record Files

absExit

```
absExit(
     )
     => none
```

Description

Closes Abstract Generator.

Arguments

None

Value Returned

None

Options

`SortOrder { Ascending | Descending }`

Specifies the sort order to apply to cells. The default is `Ascending`.

`SortView`
`{ Layout | Logical | Pins | Extract | Abstract | Verify }`

Sort cells based on the presence of a named view. The default is `Abstract`.

`SortMethod { None | Name | Height | Status | Selection }`

Specifies the sort method. The default is `Name`.

Examples

Specifies that those cells in the current bin that have a `Pins` view present are sorted by `Name` in `Ascending` order.

```
absSetOption "SortOrder" "Ascending"
=> 1
absSetOption "SortView" "Pins"
=> 1
absSetOption "SortMethod "Name"
```

```
=> 1
absSort
=> t
```

Related Topics

Sorting Cells in Abstract Generator

absRevalidateSelectedCells

```
absRevalidateSelectedCells(
     )
     => 0 / 1
```

Description

Revalidates all invalid views for cells selected in the Cell pane. If the view has an error or warning, this remains. The view status is no longer grayed out.

Arguments

None

Value Returned

`1`	Indicates successful operation.
`0`	Indicates an error.

Examples

Selects a cell called `cell_1` and revalidates the views present for that cell.

```
absSetOption "SelectName" ""cell_1"
=> 1
absSelect
=> 1
absRevalidateSelectedCells
=> 1
```

Related Topics

Revalidation of Cells with Invalid Views in Abstract Generator

absSort

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

Description

Sets sort criteria for those cells in the current bin.

Arguments

None

Value Returned

`t`	The function executes successfully.
`nil`	An error is encountered.

Related Topics

Sorting Cells in Abstract Generator

absVersion

```
absVersion(
     )
     => string
```

Description

Returns version, build, and operating system information.

Arguments

None

Value Returned

string	Returns the version information.

Related Topics

Launching the Standalone Abstract Generator

iagCreateMenuItem

```
iagCreateMenuItem(
     menuName
     menuItemName
     functionName
     )
     => t / nil
```

Description

Creates a custom menu item with the user-specified name and adds it to the integrated Abstract Generator. The menu is created if it is not already available, and the menu item is added under it. Function name is the callback procedure for the menu item.

Arguments

menuName	Specifies the name of the menu under which the menu item is created. For example, Tools, Custom_Hook, etc.
menuItemName	Specifies the name of the menu item. For example, Save, Open, etc.
functionName	Specifies the name of the function that can be run using the menu item. *functionName* can be passed as a string or a symbol.

Value Returned

t	The menu item is created.
nil	No menu item is created.

Examples

Creates a menu item named `Open Pre_Hook` under the menu `Custom_Menu` and registers the procedure call `displayForm` on clicking the menu item.

```
iagCreateMenuItem("Custom_Menu" "Open Pre_Hook" "displayForm")
```

Related Topics

Integrated Abstract Generator

iagGenAbstract

```
iagGenAbstract(
     t_libName
     l_cellNames
     t_viewName
     t_binName
     t_optionsFilePath
     )
     => t / nil
```

Description

Runs Integrated Abstract Generator on the specified cellviews with the given settings.

Arguments

t_libName	Name of the library in which the cellviews reside.
l_cellNames	List of cells for which abstracts are to be generated.
t_viewName	View name for which abstracts are to be generated.
t_binName	Bin in which the cells are to be placed. Valid values: `"Core"`, `"Block"`, `"IO"`, and `"Corner"`
t_optionsFilePath	Absolute path to an options file that defines the abstract generation rules.

Value Returned

`t`	The abstracts were generated.
`nil`	The abstracts could not be generated because of an error while running the tool. The error message is displayed separately.

Examples

Generates abstracts for the `layout_merge` view of cells `merge1` and `merge2` in the `test` library. The settings in the specified options file are applied and the cells are placed in the `Core` bin.

```
iagGenAbstract("test" list("merge1" "merge2") "layout_merge" "Core" "/home/user2/
.abstract.options")
```

Related Topics

Integrated Abstract Generator

10

Configure Physical Hierarchy Functions

This topic provides a list of Cadence® SKILL functions associated with the Configure Physical Hierarchy functionality in the Virtuoso® Layout Suite XL layout editor (Layout XL).

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Uprev Functions

Use the functions described in this section to convert existing designs and libraries to use the IC 6.1 Layout XL schema.

cphUprevDesign

cphUprevIncremental

cphUprevLibrary

cphUprevDesign

```
cphUprevDesign(
     t_physLib
     t_physCell
     t_physView
     t_cfgLib
     t_cfgCell
     t_cfgView
     )
     => t / nil
```

Description

Converts the specified design to use a physical configuration view. You must specify the name of the design to be converted and the name of the physical configuration to use.

Arguments

t_physLib	Name of the library containing the physical cell.
t_physCell	Name of the cell containing the physical view to be converted.
t_physView	Name of the physical view to be converted.
t_cfgLib	Name of the library in which the physical configuration cellview is to be created.
t_cfgCell	Name of the cell in which the physical configuration cellview is to be created.
t_cfgView	Name of the physical configuration view.

Value Returned

`t`	The design was converted.
`nil`	The design was not converted.

Example

Converts design `libA/cellA/layout` to use a physical configuration view called `physConfig` in the same library and cell directory.

```
cphUprevDesign( "libA" "cellA" "layout" "libA" "cellA" "physConfig" )
```

cphUprevIncremental

```
cphUprevIncremental(
     t_libName
     t_cellName
     t_physConfigView
     )
     => t / nil
```

Description

Modify existing physical configuration views for the library, cell and/or physical configuration names provided by cleaning up the data related to LAM physical stop list.

Arguments

t_libName	Name of the library to uprev physical configuration views.
t_cellName	Name of the cell in the library to uprev physical configuration views.
t_physConfigView	Name of the physical configuration view to be upreved.

Value Returned

`t`	The design was converted.
`nil`	The design was not converted.

Example

This example converts design `libA/cellA/layout` to use a physical configuration view called `physConfigView` in the same library and cell directory.

```
cphUprevIncremental("lib")
cphUprevIncremental("lib" "cellA")
cphUprevIncremental("lib" "cellA" "physConfig")
```

cphUprevLibrary

```
cphUprevLibrary(
     t_physLib
     [ t_cfgBaseName ]
     )
     => t / nil
```

Description

Converts the specified library to use physical configuration views. You must specify the name of the library to be converted. You can optionally specify a name for the configuration view that is generated (the default is `physConfig`).

Arguments

t_physLib	Name of the library to be converted.

t_cfgBaseName	Base name for the configuration view to be generated. The default is `physConfig`. If there are multiple schematic views to be converted, the base name is prepended to each schematic view name.
	For example, if your cell has the following views before conversion:
	`schematic`
	`schematic1`
	`schematic2`
	Then it will have the following views after conversion:
	`physConfig`
	`physConfig_schematic1`
	`physConfig_schematic2`
	`schematic`
	`schematic1`
	`schematic2`

Value Returned

`t`	The library was converted.
`nil`	The library was not converted.

Examples

This example converts library `libA` to use a physical configuration view with the default name `physConfig`.

```
cphUprevLibrary( "libA" )
```

This example converts library `libA` to use a physical configuration view called `pConfig`.

```
cphUprevLibrary("libA" "pConfig")
```

Access Functions

Use the functions described in this section to create, open, and save physical configuration views, and to get information associated with a physical configuration view.

cphChangeEditMode

cphCloseConfig

cphCloseWindow

cphCreatePhysConfig

cphFindOpenConfig

cphGetAllOpenConfigs

cphGetCell

cphGetLayoutXLConfig

cphGetLib

cphGetPhysicalTermName

cphGetSelectedSet

cphGetView

cphGetWinConfig

cphGetWindowId

cphIsConfigModified

cphIsLeaf

cphIsReadOnly

cphIsRemoveDevice

cphIsUnderPhysConfig

cphLaunchFromLayout

cphOpenConfig

cphChangeEditMode

```
cphChangeEditMode(
     g_cphID
     t_newMode
     )
     => t / nil
```

Description

Changes the physConfig from edit mode to the specified mode. If switching to read mode, `r`, and the physConfig has changed since the last edit, a prompt for saving or discarding the edits displays.

Arguments

g_cphID	ID of physical configuration view.
t_newMode	The mode in which the physConfig open for editing needs to be reopened. Other supported modes are: ■ Read mode (`r`) ■ Access mode (`a`)

Value Returned

`t`	Successfully changed the edit mode of the physical configuration view.
`nil`	The edit mode of the physical configuration view could not be changed.

Example

For the specified layout cellview ID, the physical configuration view is changed to *Read mode* (`r`).

```
cph = cphGetLayoutXLConfig(layoutCVId)
cphChangeEditMode(cph "r")
```

cphCloseConfig

```
cphCloseConfig(
     g_physConfigID
     )
     => t / nil
```

Description

Closes the physical configuration view associated with the specified ID. The physical configuration view is not saved.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t	The physical configuration view was closed.
nil	The physical configuration view was not closed.

Example

```
cphCloseConfig(physConfigID)
```

cphCloseWindow

```
cphCloseWindow(
     g_physConfigID
     )
     => t / nil
```

Description

Closes the graphical user interface for the physical configuration view associated with the specified ID. The physical configuration view is not saved.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t	The graphical user interface was closed.
nil	The graphical user interface was not closed.

Example

```
cphCloseWindow(physConfigID)
```

cphCreatePhysConfig

```
cphCreatePhysConfig(
     t_cfgLib
     t_cfgCell
     t_cfgView
     t_logLib
     t_logCell
     t_logView
     [ ?libList t_physLibList ]
     [ ?switchList t_switchList ]
     [ ?stopList t_stopList ]
     [ ?cstList t_cstList ]
     )
     => g_physConfigID / nil
```

Description

Creates a physical configuration view based on a specified logical view and returns the associated physical configuration ID. If the physical configuration view exists already, the system issues a warning.

Arguments

t_cfgLib

Name of the library in which the physical configuration cellview is to be created.

t_cfgCell

Name of the cell in which the physical configuration cellview is to be created.

t_cfgView

Name of the physical configuration view to create.

t_logLib

Name of the library containing the logical view.

t_logCell

Name of the cell containing the logical view.

t_logView

Name of the logical view.

?libList "*t_physLibList*"

List of physical library names. These libraries are searched to find the corresponding physical cell for a given logical cell.

Separate each name with a space and enclose the list in quotation marks.

?switchList "*t_switchList*"

List of logical view names used to descend into a hierarchical design.

Separate each name with a space and enclose the list in quotation marks.

?stopList "*t_stopList*"

List of physical view names used to determine the corresponding physical view for a given logical view.

When traversing a hierarchy, Configure Physical Hierarchy stops when it encounters a view with one of the specified names.

Separate each name with a space and enclose the list in quotation marks.

`?cstList "`*`t_cstList`*`"`

List of constraint view names. This list is passed to the Virtuoso Schematic Editor when it is opened in the context of a physical configuration, allowing it to determine whether the contents of the Constraint Manager must be updated.

Separate each name with a space and enclose the list in quotation marks.

Value Returned

`g_physConfigID`	ID of the physical configuration cellview.
`nil`	The new physical configuration view was not created.

Example

```
physConfigID=cphCreatePhysConfig("cph" "TopCell" "cph" "physConfig" "TopCell"
"schematic" ?libList "cph" ?switchList "physConfig schematic symbol"
?stopList "layout abstract")
```

cphFindOpenConfig

```
cphFindOpenConfig(
     t_cfgLib
     t_cfgCell
     t_cfgView
     )
     => g_physConfigID / nil
```

Description

Returns the ID associated with the specified physical configuration view, if it exists and is already open.

Arguments

t_cfgLib	Name of the library containing the physical configuration cellview.
t_cfgCell	Name of the cell containing the physical configuration cellview.
t_cfgView	Name of the physical configuration view.

Value Returned

g_physConfigID	ID of the physical configuration cellview.
nil	The physical configuration view was not opened.

Example

```
cphFindOpenConfig("cph" "TopCell" "physconfig")
```

cphGetAllOpenConfigs

```
cphGetAllOpenConfigs(
     )
     => l_physConfigID / nil
```

Description

Returns the IDs associated with all the open physical configuration views.

Arguments

None

Value Returned

l_physConfigID	List of IDs of the physical configuration cellviews currently open.
nil	No physical configuration views are currently open.

Example

Finds all the open physical configuration cellviews and makes them read only.

```
foreach(cph cphGetAllOpenConfigs() cphChangeEditMode(cph "r"))
```

cphGetCell

```
cphGetCell(
     g_physConfigID
     )
     => t_cellName / nil
```

Description

Returns the name of the cell containing the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t_cellName	Name of the cell containing the specified physical configuration view.
nil	The command was unsuccessful.

Example

```
cphGetCell(physConfigID)
```

Related Topics

cphGetLayoutXLConfig
cphGetLib
cphGetView

cphGetLayoutXLConfig

```
cphGetLayoutXLConfig(
     d_physCellviewID
     )
     => g_physConfigID / nil
```

Description

Returns the ID of the physical configuration cellview associated with a specified physical cellview ID.

Arguments

d_physCellviewID	Database ID of the physical cellview.

Value Returned

g_physConfigID	ID of the physical configuration cellview.
nil	The command was unsuccessful.

Example

```
physConfigID=cphGetLayoutXLConfig(layoutId)
```

Related Topics

cphGetLib
cphGetCell
cphGetView

cphGetLib

```
cphGetLib(
     g_physConfigID
     )
     => t_libName / nil
```

Description

Returns the name of the library containing the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t_libName	Name of the library containing the physical configuration view.
nil	The command was unsuccessful.

Example

```
cphGetLib(physConfigID)
```

Related Topics

cphGetLayoutXLConfig
cphGetCell
cphGetView

cphGetPhysicalTermName

```
cphGetPhysicalTermName(
     g_physConfigID
     t_path
     t_logTermName
     )
     => t_physTermName / nil
```

Description

Returns the name of the corresponding physical terminal name for a specified logical terminal on a specified occurrence.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_logTermName	Name of the logical terminal.

Value Returned

t_physTermName	Name of the physical terminal.
nil	There is no mapping information for the specified logical terminal name or the logical terminal name is invalid.

Example

```
physTermName=cphGetPhysicalTermName(physConfigID "I0/N0" "S")
```

cphGetSelectedSet

```
cphGetSelectedSet(
     g_physConfigID
     )
     => l_listOfPaths / nil
```

Description

Returns a list of paths to the selected instances in the graphical user interface of physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

l_listOfPaths	The list of paths to the selected instances.
nil	No instances are selected.

Example

```
cphGetSelectedSet(physConfigID)
    => ("I0" "I0/N0" "I2")
```

cphGetView

```
cphGetView(
     g_physConfigID
     )
     => t_viewName / nil
```

Description

Returns the view name for the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t_viewName	View name of the physical configuration view.
nil	The command was unsuccessful.

Example

```
cphGetView(physConfigID)
```

Related Topics

cphGetLayoutXLConfig
cphGetCell
cphGetLib

cphGetWinConfig

```
cphGetWinConfig(
     w_windowID
     )
     => g_physConfigID / nil
```

Description

Returns the ID of the physical configuration view associated with a specified Configure Physical Hierarchy window.

Arguments

w_windowID	ID of a Configure Physical Hierarchy window containing a physical configuration view.

Value Returned

g_physConfigID	ID of the physical configuration cellview.
nil	The command was unsuccessful.

Example

This example retrieves the physical configuration ID for the current Configure Physical Hierarchy window and sets the physical stop view list for the physical configuration view associated with that ID.

```
let( (win cph)
          win = hiGetCurrentWindow();  assuming CPH is the current window
          if(win then
             cph = cphGetWinConfig(win)
             if(cph then
                  cphSetStopList(cph "abstract layout")
             )
          )
     )
```

cphGetWindowId

```
cphGetWindowId(
     g_physConfigID
     )
     => w_windowID / nil
```

Description

Returns the window ID of the Configure Physical Hierarchy window that contains the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

w_windowID	ID of the Configure Physical Hierarchy window containing the physical configuration view.
nil	Cannot find the window ID or the window does not exist.

Example

```
cphGetWindowId(physConfigID)
```

cphIsConfigModified

```
cphIsConfigModified(
     g_physConfigID
     )
     => t / nil
```

Description

Returns whether or not the physical configuration view associated with the given ID has been modified since it was last saved.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

`t`	The physical configuration view has been modified.
`nil`	The physical configuration view has not been modified.

Example

```
cphIsConfigModified(physConfigID)
```

cphIsLeaf

```
cphIsLeaf(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Determines whether the specified occurrence is a leaf. An occurrence is considered a leaf if no hierarchy will be generated underneath it.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The specified occurrence is a leaf.
nil	The specified occurrence is not a leaf.

Example

```
cphIsLeaf(physConfigID "I0/N0")
```

cphIsReadOnly

```
cphIsReadOnly(
     g_cphID
     )
     => t / nil
```

Description

Checks whether the physical configuration view is read only.

Arguments

g_cphID	ID of the physical configuration cellview.

Value Returned

t	The physical configuration view is read only.
nil	The physical configuration view is editable, not read only.

Example

```
cph = cphGetLayoutXLConfig(layoutCVId)
cphIsReadOnly(cph)
```

cphIsRemoveDevice

```
cphIsRemoveDevice(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Determines whether there is a *Remove* rule defined for the specified occurrence in the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	There is a *Remove* rule defined for the specified occurrence.
`nil`	There is no *Remove* rule defined for the specified occurrence.

Example

```
cphIsRemoveDevice(physConfigID "I0/N0")
```

cphIsUnderPhysConfig

```
cphIsUnderPhysConfig(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Determines whether the specified occurrence exists under a sub physConfig.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The specified occurrence exists under a sub physConfig.
nil	The specified occurrence does not exist under a sub physConfig.

Example

```
cphIsUnderPhysConfig(physConfigID "I0/N0")
```

cphLaunchFromLayout

```
cphLaunchFromLayout(
     d_physCellviewID
     )
     => t / nil
```

Description

Launches the Configure Physical Hierarchy window and loads the physical configuration view associated with the specified physical cellview ID.

Arguments

d_physCellviewID	Database ID of the physical cellview.

Value Returned

t	The physical configuration view was opened in the graphical user interface.
nil	The physical configuration view was not opened.

Example

```
cphLaunchFromLayout(physCellviewID)
```

cphOpenConfig

```
cphOpenConfig(
     t_cfgLib
     t_cfgCell
     t_cfgView
     [ t_mode { r | a } ]
     )
     => g_physConfigID / nil
```

Description

Opens the specified physical configuration view by name in either read (the default) or append mode and returns its ID.

Arguments

t_cfgLib	Name of the library containing the physical configuration cellview.
t_cfgCell	Name of the cell containing the physical configuration cellview.
t_cfgView	Name of the physical configuration view to open.
t_mode	Mode in which the physical configuration view is opened: either read or append. The default is `r`. Valid values: `r`, `a`

Value Returned

g_physConfigID	ID of the physical configuration cellview.
`nil`	The physical configuration view was not opened.

Example

```
physConfigID=cphOpenConfig("cph" "TopCell" "physConfig" "a")
```

cphOpenWindow

```
cphOpenWindow(
     g_physConfigID
     )
     => t / nil
```

Description

Opens the graphical user interface and displays the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t	The physical configuration view was opened in the graphical user interface.
nil	The physical configuration view was not opened in the graphical user interface.

Example

```
cphOpenWindow(physConfigID)
```

cphReplaceCellName

```
cphReplaceCellName(
     t_oldName
     t_newName
     ?cfgLib t_libName
     [ ?cfgCell t_cellName ]
     [ ?cfgView t_viewName ]
     )
     => t / nil
```

Description

Enables update or repair of physConfig views after performing linux copy to copy the library files instead of using the Library Manager *Copy Library* option.

Arguments

t_oldName	Original name of the cell.
t_newName	New name of the cell.
?cfgLib *t_libName*	Name of the library that contains the physConfig views to be updated.
?cfgCell *t_cellName*	Name of the cell that contains the physConfig views to be updated.
?cfgView *t_viewName*	Name of the physConfig view to be updated.

Value Returned

t	Name of the cell replaced with the new name.
nil	The command was unsuccessful.

Example

Copies a cell for one location to a new location.

```
cp -R cph/nand cphNew/nandNew
```

Finds all physConfigs in library "cphNew" and replaces any cell called "nand" with "nandNew" because only ?cfgLib is specified.

```
cphReplaceCellName("nand" "nandNew" ?cfgLib "cphNew")
```

Find all physConfigs in cell "cphNew/nandNew" and replaces any cell called "nand" with "nandNew" because both ?cfgLib and ?cfgCell are specified.

```
cphReplaceCellName("nand" "nandNew" ?cfgLib "cphNew" ?cfgCell "nandNew")
```

Finds the specified physConfig and replaces any cell called "nand" with "nandNew" because ?cfgLib, ?cfgCell, and ?cfgView are specified.

```
cphReplaceCellName("nand" "nandNew" ?cfgLib "cphNew" ?cfgCell "nandNew" ?cfgView
"physConfig")
```

cphReplaceLibName

```
cphReplaceLibName(
     t_oldName
     t_newName
     ?cfgLib t_libName
     [ ?cfgCell t_cellName ]
     [ ?cfgView t_viewName ]
     )
     => t / nil
```

Description

Enables update or repair of physConfig views after performing linux copy to copy the library files instead of using the Library Manager *Copy Library* option.

Arguments

t_oldName	Original name of the library.
t_newName	New name of the library.
?cfgLib *t_libName*	Name of the library that contains the physConfigs to be updated.
?cfgCell *t_cellName*	Name of the cell that contains the physConfig views to be updated.
?cfgView *t_viewName*	Name of the physConfig view to be updated.

Value Returned

t	Name of the library replaced with the new name.
nil	The command was unsuccessful.

Examples

User copies the library called "cph" to "cphNew".

```
cp -R cph cphNew
```

User updates the `cds.lib` and launches Virtuoso. To refresh and reflect the edits made to cds.lib, the user refreshes or runs the SKILL function `ddUpdateLibList()`. After Virtuoso is refreshed, the user can run `cphReplaceLibName` to find all the physConfigs in library "`cphNew`" and replace any library named "`cph`" with "`cphNew`".

```
cphReplaceLibName("cph" "cphNew" ?cfgLib "cphNew")
```

cphReplaceViewName

```
cphReplaceViewName(
     t_oldName
     t_newName
     ?cfgLib t_libName
     [ ?cfgCell t_cellName ]
     [ ?cfgView t_viewName ]
     )
     => t / nil
```

Description

Enables update or repair of physConfig views after performing linux copy to copy the library files instead of using the Library Manager *Copy Library* option.

Arguments

t_oldName	Original name of the view.
t_newName	New name of the view.
?cfgLib *t_libName*	Name of the library that contains the physConfigs to be updated.
?cfgCell *t_cellName*	Name of the cell that contains the physConfig views to be updated.
?cfgView *t_viewName*	Name of the physConfig view to be updated.

Value Returned

t	Name of the physConfig view replaced with the new name.
nil	The command was unsuccessful.

Example

Copies a view for one location to a new location with a new name.

```
cp -R cph/nand/schematic cphNew/nand/sch
```

Finds all physConfigs in cell "cphNew/nand" and replaces any view called "schematic" with "sch" because ?cfgLib and ?cfgCell are specified.

```
cphReplaceViewName("schematic" "sch" ?cfgLib "cphNew" ?cfgCell "nand")
```

Finds the specified physConfig and replaces any view called "schematic" with "sch" because ?cfgLib, ?cfgCell and ?cfgView are specified.

```
cphReplaceViewName("schematic" "sch" ?cfgLib "cphNew" ?cfgCell "nand" ?cfgView
"physConfig")
```

cphSaveAsConfig

```
cphSaveAsConfig(
     g_physConfigID
     t_cfgLib
     t_cfgCell
     t_cfgView
     )
     => t / nil
```

Description

Saves the physical configuration view associated with an ID under the library, cell, and view names you specify.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_cfgLib	Name of the library in which the physical configuration cellview is to be saved.
t_cfgCell	Name of the cell in which the physical configuration cellview is to be saved.
t_cfgView	Name for the physical configuration view.

Value Returned

t	The physical configuration view was saved.
nil	The physical configuration view was not saved.

Example

```
cphSaveAsConfig(physConfigID "cph" "TopCell" "physConfig")
```

cphSaveConfig

```
cphSaveConfig(
     g_physConfigID
     )
     => t / nil
```

Description

Saves the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

`t`	The physical configuration view was saved.
`nil`	The physical configuration view was not saved.

Example

```
cphSaveConfig(physConfigID)
```

Top Cell Pane Function

Use the functions in this section to get the database ID of the top schematic cellview associated with a specified physical configuration and the names of associated libraries, cells, and views.

cphGetTopCellName

cphGetTopCellView

cphGetTopLibName

cphGetTopViewName

Related Topics

Top Cell

cphGetTopCellName

```
cphGetTopCellName
     g_cphManager
     )
     => t_cellName
```

Description

Returns the cell name of the top schematic cellview in the specified physical configuration view.

Arguments

g_cphManager	CPH manager ID of the associated physical configuration view.

Value Returned

t_cellName	Cell name of the top schematic cellview.

Example

```
cph = cphOpenConfig("MOSLIB" "NAND" "physConfig")
cphGetTopCellName(cph) => "NAND"
```

cphGetTopCellView

```
cphGetTopCellView(
     g_physConfigID
     )
     => d_logCellviewID / nil
```

Description

Returns the database ID of the logical cellview associated with the specified physical configuration ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

d_logCellviewID	Database ID of the logical cellview.
nil	The function was unsuccessful.

Example

```
logCellviewID=cphGetTopCellView(physConfigID)
```

cphGetTopLibName

```
cphGetTopLibName(
     g_cphManager
     )
     => t_libName
```

Description

Returns the library name of the top schematic cellview in the specified physical configuration view.

Arguments

g_cphManager	CPH manager ID of the associated physical configuration view.

Value Returned

t_libName	Library name of the top schematic cellview.

Example

```
cph = cphOpenConfig("MOSLIB" "NAND" "physConfig")
cphGetTopLibName(cph) => "MOSLIB"
```

cphGetTopViewName

```
cphGetTopViewName(
     g_cphManager
     )
     => t_viewName
```

Description

Returns the view name of the top schematic cellview in the specified physical configuration view.

Arguments

g_cphManager	CPH manager ID of the associated physical configuration view.

Value Returned

t_viewName	View name of the top schematic cellview.

Example

```
cph = cphOpenConfig("MOSLIB" "NAND" "physConfig")
cphGetTopViewName(cph) => "schematic"
```

Global Bindings Pane Functions

Use the functions described in this section to set and retrieve the values for the view lists set in the Configure Physical Hierarchy Global Bindings pane.

cphGetCstList

cphGetLibList

cphGetStopList

cphGetViewList

cphSetCstList

cphSetLibList

cphSetStopList

cphSetViewList

Related Topics

Global Bindings

cphGetCstList

```
cphGetCstList(
     g_physConfigID
     )
     => t_cstList / nil
```

Description

Returns the names of views containing constraint data for the physical configuration view associated with the specified ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t_cstList	List of constraint view names.
nil	The constraint view list was not returned.

Example

```
cstList=cphGetCstList(physConfigID)
```

cphGetLibList

```
cphGetLibList(
     g_physConfigID
     )
     => t_physLibList / nil
```

Description

Returns the list of physical library names list for the physical configuration view associated with the specified ID. These are the libraries that are searched to find the corresponding physical cell for a given logical cell.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t_physLibList	List of physical library names.
nil	The physical library list was not returned.

Example

```
physLibList=cphGetLibList(physConfigID)
```

cphGetStopList

```
cphGetStopList(
     g_physConfigID
     [ t_path ]
     )
     => t_stopList / nil
```

Description

Returns the physical stop view list defined for the physical configuration view associated with the specified ID or, if you specify the path to an occurrence, the effective physical stop view list used by that occurrence.

The physical stop view list specifies the view names used to determine the corresponding physical view for a given logical view. When traversing a hierarchy, Configure Physical Hierarchy uses the first view it encounters with one of the specified names.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance. **Note:** Although it is optional, this argument requires no keyword.

Value Returned

t_stopList	List of physical view names.
nil	The physical stop view list was not returned.

Example

This example returns the physical stop view list defined for the specified physical configuration view.

```
stopList=cphGetStopList(physConfigID)
```

This example returns the effective physical stop view list used by the specified occurrence.

```
stopList=cphGetStopList(physConfigID "I0/N0")
```

cphGetViewList

```
cphGetViewList(
     g_physConfigID
     [ t_path ]
     )
     => t_switchList / nil
```

Description

Returns the logical switch view list for the physical configuration view associated with the specified ID or, if you specify the path to an occurrence, the effective logical switch view list used by that occurrence.

The list specifies the view names used to descend into a hierarchical design to find schematic views.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance. **Note:** Although it is optional, this argument requires no keyword.

Value Returned

t_switchList	List of logical view names.
nil	The logical switch view list was not returned.

Example

This example returns the logical switch view list defined for the specified physical configuration view.

```
switchList=cphGetViewList(physConfigID)
```

This example returns the effective logical switch view list used by the specified occurrence.

```
switchList=cphGetViewList(physConfigID "I0/N0")
```

cphSetCstList

```
cphSetCstList(
     g_physConfigID
     t_cstList
     )
     => t / nil
```

Description

Sets the constraint view list for the physical configuration view associated with the specified ID. This list specifies the names of the views containing constraint data.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_cstList	List of constraint view names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The constraint view list was set.
nil	The constraint view list was not set.

Example

```
cphSetCstList(physConfigID "constraint cstView")
```

cphSetLibList

```
cphSetLibList(
     g_physConfigID
     t_physLibList
     )
     => t / nil
```

Description

Sets the physical library list for the physical configuration view associated with the specified ID. This list specifies the names of the libraries searched to find the corresponding physical cell for a given logical cell.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_physLibList	List of physical library names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The physical library list was set.
nil	The physical library list was not set.

Example

```
cphSetLibList(physConfigID "phy1 cph")
```

cphSetStopList

```
cphSetStopList(
     g_physConfigID
     t_stopList
     )
     => t / nil
```

Description

Sets the physical stop view list for the physical configuration view associated with the specified ID. The list specifies the view names that are used to determine the corresponding physical view for a given logical view. When traversing a hierarchy, Configure Physical Hierarchy uses the first view it encounters with one of the specified names.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_stopList	List of physical view names used to determine the corresponding physical view for a given logical view. When traversing a hierarchy, Configure Physical Hierarchy stops when it encounters a view with one of the specified names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

`t`	The physical stop view list was set.
`nil`	The physical stop view list was not set.

Example

This example retrieves the physical configuration ID for the current Configure Physical Hierarchy window and sets the physical stop view list for the physical configuration view associated with that ID.

```
let( (win cph)
          win = hiGetCurrentWindow();  assuming CPH is the current window
          if(win then
             cph = cphGetWinConfig(win)
```

```
            if(cph then
                 cphSetStopList(cph "abstract layout")
            )
       )
  )
```

cphSetViewList

```
cphSetViewList(
     g_physConfigID
     t_switchList
     )
     => t / nil
```

Description

Sets the logical switch view list for the physical configuration view associated with the specified ID. The list specifies the view names used to descend into a hierarchical design to find schematic views.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_switchList	List of logical view names used to descend into a hierarchical design. Separate each name with a space and enclose the list in quotation marks.

Value Returned

`t`	The logical switch view list was set.
`nil`	The logical switch view list was not set.

Example

```
cphSetViewList(physConfigID "schematic symbol")
```

Tree Pane Functions

Use the functions described in this section to set and retrieve view lists, bindings, and force descend values for cells, instances, and occurrences in the hierarchy configuration tables.

cphClearCellPhysicalBinding

cphClearInstPhysicalBinding

cphClearOccurPhysicalBinding

cphClearSchCellPhysicalBinding

cphClearSchInstPhysicalBinding

cphGetCellForceDescend

cphGetCellPhysicalBinding

cphGetCellPhysicalCell

cphGetCellPhysicalLib

cphGetCellPhysicalView

cphGetCellStopList

cphGetCellViewBinding

cphGetCellViewList

cphGetForceDescend

cphGetInstForceDescend

cphGetSchCellPhysicalBinding

cphGetSchInstForceDescend

cphGetSchInstPhysicalBinding

cphGetInstPhysicalCell

cphGetInstPhysicalLib

cphGetInstPhysicalView

cphSetSchCellPhysicalBinding

cphSetSchInstPhysicalBinding

cphSetInstStopList

cphSetInstStopPoint

cphSetInstViewBinding

cphSetInstViewList

cphSetOccurForceDescend

cphSetOccurPhysicalBinding

cphSetOccurStopList

cphSetOccurStopPoint

cphSetOccurViewBinding

cphSetOccurViewList

Related Topics

Hierarchy Configuration Mode in the CPH Window

cphClearCellPhysicalBinding

```
cphClearCellPhysicalBinding(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Removes the physical binding for a logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.

Value Returned

t	The physical binding was removed.
nil	The physical binding was not removed.

Example

```
cphClearCellPhysicalBinding(physConfigID "cph" "nand")
```

cphClearInstPhysicalBinding

```
cphClearInstPhysicalBinding(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Removes the physical binding for a specific instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The physical binding was removed.
nil	The physical binding was not removed.

Example

```
cphClearInstPhysicalBinding(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphClearOccurPhysicalBinding

```
cphClearOccurPhysicalBinding(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Removes the physical binding for a specific occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The physical binding was removed.
`nil`	The physical binding was not removed.

Example

```
cphClearOccurPhysicalBinding(physConfigID "I0/N0")
```

cphClearSchCellPhysicalBinding

```
cphClearSchCellPhysicalBinding(
     d_topSchCVID
     t_libName
     t_cellName
     [ ?layCVId d_layCVId ]
     )
     => t / nil
```

Description

Removes the physical binding for the cell specified using the top-level schematic and layout cellview IDs instead of the cphManager ID. After the physical binding for the specified cell is removed, the update is also saved to the physConfig view.

Arguments

d_topSchCVID	Database ID of the top-level schematic cellview.
t_libName	Name of the library containing the cell to be updated.
t_cellName	Name of the cell to be updated.
?layCVId *d_layCVId*	Database ID of the top-level layout cellview associated with the specified cell.

Value Returned

t	The physical binding was removed.
nil	The physical binding was not removed.

Example

Removes the physical binding of the specified cell, `nand`.

```
scv -> Schematic cellview ID of the top-level schematic in a VLS XL session
lcv -> Layout cellview ID of the top-level layout in a VLS XL session
cphClearSchCellPhysicalBinding(scv "cph" "nand" ?layCVId lcv) => t
```

cphClearSchInstPhysicalBinding

```
cphClearSchInstPhysicalBinding(
     d_topSchCVID
     d_schInstID
     g_all
     [ ?layCVId d_layCVId ])
     => t / nil
```

Description

Removes the physical binding for the instance specified using the top-level schematic and layout cellview IDs instead of the cphManager ID. After the physical binding for the specified instance is removed, the update is also saved to the physConfig view.

Arguments

d_topSchCVID	Database ID of the top-level schematic cellview.
d_schInstID	Database ID of the schematic instance for which physical binding needs to be removed.
g_all	If set to `t`, updates the sub level physConfig views, if the specified instance ID is inside a sub level physConfig. If set to `nil`, updates no sub level physConfigs.
`?layCVId` *d_layCVId*	Database ID of the top-level layout cellview associated with the specified schematic instance.

Value Returned

`t`	The physical binding was removed.
`nil`	The physical binding was not removed.

Example

Removes the physical binding on instance `N0`, which is instantiated in schematic cellview `nand2_A` using a Layout XL session that is represented by `scv` and `lcv`.

```
scv -> top-level schematic cellview ID in a VLS XL -XL session
lcv  -> top level layout cellview id in the VXL-XL session
```

```
i0   -> Instance id for instance "I0" in the top level schematic cellview
n0  -> Instance id for instance"N0" in lower level nand2_A schematic cellview
cphClearSchInstPhysicalBinding(scv n0 t ?layCVId lcv)=> t
```

cphGetCellForceDescend

```
cphGetCellForceDescend(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns whether the force descend attribute is set for a specific cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The force descend is set for the specified cell.
nil	The force descend is not set for the specified cell.

Example

```
cphGetCellForceDescend(cph "cph" "nand") => t
```

cphGetCellPhysicalBinding

```
cphGetCellPhysicalBinding(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_physicalBinding / nil
```

Description

Returns the physical library, cell, and view mapped to a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical cell.

Value Returned

t_physicalBinding	Name of the physical library, cell, and view set on the specified cell.
nil	The physical binding is not set on the specified cell.

Example

```
cphGetCellPhysicalBinding(physConfigID "cph" "nand2")
```

cphGetCellPhysicalCell

```
cphGetCellPhysicalCell(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_physCell / nil
```

Description

Returns the name of the physical cell mapped to a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_physCell	Name of the physical cell.
nil	The physical cell name was not returned.

Example

```
cphGetCellPhysicalCell(physConfigID "cph" "nand")
```

cphGetCellPhysicalLib

```
cphGetCellPhysicalLib(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_physLib / nil
```

Description

Returns the physical library mapped to a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical cell.

Value Returned

t_physLib	Name of the physical library.
nil	The physical library name was not returned.

Example

```
cphGetCellPhysicalLib(physConfigID "cph" "nand")
```

cphGetCellPhysicalView

```
cphGetCellPhysicalView(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_physView / nil
```

Description

Returns the physical view mapped to a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical cell.

Value Returned

t_physView	Name of the physical view.
nil	The physical view name was not returned.

Example

```
cphGetCellPhysicalView(physConfigID "cph" "nand")
```

cphGetCellStopList

```
cphGetCellStopList(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_stopList / nil
```

Description

Returns the inherited physical stop view list for a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_stopList	List of physical view names.
nil	The stop list was not returned.

Example

```
stopList=cphGetCellStopList(physConfigID "cph" "nand")
```

cphGetCellViewBinding

```
cphGetCellViewBinding(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_logViewToUse / nil
```

Description

Returns the view to use for a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.

Value Returned

t_logViewToUse	Name of the logical view to use.
nil	The cellview binding was not returned.

Example

```
logViewToUse=cphGetCellViewBinding(physConfigID "cph" "nand")
```

cphGetCellViewList

```
cphGetCellViewList(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_switchList / nil
```

Description

Returns the inherited logical switch view list for a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_switchList	List of logical view names.
nil	The logical switch view list was not returned.

Example

```
switchList=cphGetCellViewList(physConfigID "cph" "nand")
```

cphGetForceDescend

```
cphGetForceDescend(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns whether or not an effective force descend attribute is set for a specific occurrence in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The force descend is set for the specified occurrence.
nil	The force descend is not set for the specified occurrence.

Example

```
forceDescend=cphGetForceDescend(physConfigID "I0/N2")
```

cphGetInstForceDescend

```
cphGetInstForceDescend(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_path
     )
     => t / nil
```

Description

Returns whether or not the force descend attribute is set for a specific instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t	The force descend is set for the specified instance.
nil	The force descend is not set for the specified instance.

Example

```
forceDescend=cphGetInstForceDescend(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetSchCellPhysicalBinding

```
cphGetSchCellPhysicalBinding(
     d_topSchCVID
     t_libName
     t_cellName
     [ ?layCVId d_layCVId ]
     )
     => t_result / nil
```

Description

Returns the physical binding of a cell specifying the top-level schematic and layout cellview IDs instead of the cphManager ID.

Arguments

d_topSchCVID	Database ID of the top-level schematic cellview.
t_libName	Name of the library containing the schematic cell.
t_cellName	Name of the cell to be queried.
?layCVId *d_layCVId*	Database ID of the top-level layout cellview.

Value Returned

t_result	Name of the physical library, cell, and view set on the specified schematic cell.
nil	The physical binding is not set on the specified cell.

Example

Prints the physical binding set on the specified schematic cell without having to find the physConfig view and the corresponding cphManager ID.

```
scv -> Schematic cellview ID of the top-level schematic in a VLS XL session
lcv -> Layout cellview ID of the top-level layout in a VLS XL session
cphGetSchCellPhysicalBinding(scv "cph" "nand" ?layCVId lcv) => "cph nand layout"
```

cphGetSchInstForceDescend

```
cphGetSchInstForceDescend(
     d_topSchCVID
     d_schInstID
     [ ?layCVId d_layCVId ]
     )
     => t / nil
```

Description

Returns whether an instance has a force descend, specifying the top-level schematic and layout cellview IDs instead of the cphManager ID.

Arguments

d_topSchCVID	Database ID of the top-level schematic cellview.
d_schInstID	Database ID of the schematic instance.
?layCVId *d_layCVId*	Database ID of the top-level layout cellview.

Value Returned

t	The force descend is set for the specified instance.
nil	The force descend is not set for the specified instance.

Example

Instance "I0" in the top-level schematic has a force descend set.

```
scv -> top-level schematic cellview ID in a VLS XL session
lcv -> top-level layout cellview ID in a VLS XL session
i0 -> Instance ID for instance "I0" in the top-level schematic cellview
n0 -> Instance ID for instance "N0" in the lower-level nand2_A schematic cellview
cphGetSchInstForceDescend(scv i0 ?layCVId lcv) => t
```

Checks and reports if instance "N0" at the next lower level of hierarchy has a force descend.

```
cphGetSchInstForceDescend(scv n0 ?layCVId lcv) => nil
```

cphGetSchInstPhysicalBinding

```
cphGetSchInstPhysicalBinding(
     d_topSchCVID
     d_schInstID
     [ ?layCVId d_layCVId ]
     )
     => t_result / nil
```

Description

Returns the physical binding of an instance by specifying its top-level schematic and layout cellview IDs instead of specifying the cphManager ID.

Arguments

d_topSchCVID

Database ID of the top-level schematic cellview.

d_schInstID

Database ID of the schematic instance.

?layCVId *d_layCVId*

Database ID of the top-level layout cellview.

Value Returned

t_result	The physical binding for the instance specified as a string value.
nil	The command failed.

Example

Instance I0 does not have any physical binding, so the function retrieves the physical binding of the next-level instance, nmos "N0".

```
scv -> top-level schematic cellview ID in a VLS XL session
lcv -> top-level layout cellview ID in a VLS XL session
i0 -> Instance ID for instance "I0" in the top-level schematic cellview
n0 -> Instance ID for instance "N0" in the lower-level nand2_A schematic cellview
cphGetSchInstPhyiscalBinding(scv i0  ?layCVId lcv) => ""
```

```
cphGetSchInstPhyiscalBinding(scv n0 ?layCVId lcv) => "cph nmos layout"
```

cphGetInstPhysicalCell

```
cphGetInstPhysicalCell(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_physCell / nil
```

Description

Returns the name of the physical cell mapped to a specified instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_physCell	Name of the physical cell.
nil	The physical cell name was not returned.

Example

```
cphGetInstPhysicalCell(physConfigID "cph" "nand" "schematic" "I0")
```

cphGetInstPhysicalLib

```
cphGetInstPhysicalLib(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_physLib / nil
```

Description

Returns the physical library mapped to a specified instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_physLib	Name of the physical library.
nil	The physical library name was not returned.

Example

```
cphGetInstPhysicalLib(physConfigID "cph" "nand" "schematic" "I0")
```

cphGetInstPhysicalView

```
cphGetInstPhysicalView(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_physView / nil
```

Description

Returns the physical view mapped to a specified instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_physView	Name of the physical view.
nil	The physical view name was not returned.

Example

```
cphGetInstPhysicalView(physConfigID "cph" "nand" "schematic" "I0")
```

cphGetInstStopList

```
cphGetInstStopList(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_stopList / nil
```

Description

Returns the inherited stop view list for a specified instance in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_stopList	List of physical view names.
nil	The physical stop view list was not returned.

Example

```
stopList=cphGetInstStopList(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstViewBinding

```
cphGetInstViewBinding(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_logViewToUse / nil
```

Description

Returns the view to use for a specified logical instance in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_logViewToUse	Name of the logical view to use.
nil	The instance binding was not returned.

Example

```
logViewToUse=cphGetInstViewBinding(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstViewList

```
cphGetInstViewList(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_switchList / nil
```

Description

Returns the inherited logical switch view list for a specified instance in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_switchList	List of logical view names.
nil	The logical switch view list was not returned.

Example

```
switchList=cphGetInstViewList(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetOccurForceDescend

```
cphGetOccurForceDescend(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns whether or not the force descend attribute is set for a specific occurrence in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The force descend is set for the specified occurrence.
`nil`	The force descend is not set for the specified occurrence.

Example

```
forceDescend=cphGetOccurForceDescend(physConfigID "I0/N2")
```

cphGetOccurPhysicalCell

```
cphGetOccurPhysicalCell(
     g_physConfigID
     t_path
     )
     => t_physCell / nil
```

Description

Returns the physical cell binding for an occurrence of an instance in the physical configuration associated with the given ID.

Note: This function returns a physical cell name only if it is explicitly set on the occurrence. Use `cphGetPhysicalCell` to get the effective physical cell for an occurrence with no explicit setting.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_physCell	Name of the physical cell.
`nil`	There is no physical cell name set for the occurrence.

Example

This example returns the physical cell name set explicitly for an occurrence `I0/N0`.

```
physCell=cphGetOccurPhysicalCell(physConfigID "I0/N0")
```

Related Topics

cphGetPhysicalCell

cphGetOccurPhysicalLib

```
cphGetOccurPhysicalLib(
     g_physConfigID
     t_path
     )
     => t_physLib / nil
```

Description

Returns the physical library binding for an occurrence of an instance in the physical configuration associated with the given ID.

Note: This function returns a physical library name only if it is explicitly set on the occurrence. Use `cphGetPhysicalLib` to get the effective physical library name for an occurrence with no explicit setting.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_physLib	Name of the physical library.
`nil`	There is no physical library name set for the occurrence.

Example

This example returns the physical library name set explicitly for an occurrence `I0/N0`.

```
physLib=cphGetOccurPhysicalLib(physConfigID "I0/N0")
```

Related Topics

cphGetPhysicalLib

cphGetOccurPhysicalView

```
cphGetOccurPhysicalView(
     g_physConfigID
     t_path
     )
     => t_physView / nil
```

Description

Returns the view binding for an occurrence of an instance in the physical configuration associated with the given ID.

Note: This function returns a physical view name only if it is explicitly set on the occurrence. Use `cphGetPhysicalView` to get the effective physical view used by an occurrence with no explicit setting.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_physLib	Name of the physical view.
`nil`	There is no physical view name set for the occurrence.

Example

This example returns the physical view name set explicitly for an occurrence `I0/N0`.

```
physView=cphGetOccurPhysicalview(physConfigID "I0/N0")
```

Related Topics

cphGetPhysicalView

cphGetOccurStopList

```
cphGetOccurStopList(
     g_physConfigID
     t_path
     )
     => t_stopList / nil
```

Description

Returns the physical stop view list set explicitly for an occurrence of an instance in the physical configuration view associated with the given ID.

Note: This function returns the stop list only if it is explicitly set on the occurrence. Use cphGetStopList with the optional *t_path* argument to get the effective stop list for an occurrence with no explicit setting.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_stopList	List of physical view names.
nil	There is no physical stop view list set for the occurrence.

Example

This example returns the physical stop view list set explicitly on an occurrence I0/N0.

```
stopList=cphGetOccurStopList(physConfigID "I0/N0")
```

Related Topics

cphGetStopList

cphGetOccurViewBinding

```
cphGetOccurViewBinding(
     g_physConfigID
     t_path
     )
     => t_logViewToUse / nil
```

Description

Returns the logical view to use set explicitly for an occurrence of an instance in the physical configuration view associated with the given ID.

Note: This function returns the logical view to use only if it is explicitly set on the occurrence. Use `cphGetViewBinding` to get the effective stop list for an occurrence with no explicit setting.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_logViewToUse	Name of the logical view to use.
`nil`	There is no logical view to use set for the occurrence.

Example

This example returns the view to use set explicitly on an occurrence `I0/N0`.

```
logViewToUse=cphGetOccurViewBinding(physConfigID "I0/N0")
```

Related Topics

cphGetViewBinding

cphGetOccurViewList

```
cphGetOccurViewList(
     g_physConfigID
     t_path
     )
     => t_switchList / nil
```

Description

Returns the inherited logical switch view list for a specific occurrence of an instance in the physical configuration view associated with the given ID.

Note: This function returns the switch list only if it is explicitly set on the occurrence. Use `cphGetViewList` with the optional *`t_path`* argument to get the effective switch list for an occurrence with no explicit setting.

Arguments

`g_physConfigID`	ID of the physical configuration cellview.
`t_path`	Path to the specific occurrence of an instance.

Value Returned

`t_switchList`	List of logical view names.
`nil`	There is no logical switch view list set for the occurrence.

Example

This example returns the logical switch view list set explicitly on an occurrence `I0/N0`.

```
switchList=cphGetOccurViewList(physConfigID "I0/N0")
```

Related Topics

cphGetViewList

cphGetPhysicalCell

```
cphGetPhysicalCell(
     g_physConfigID
     t_path
     )
     => t_physCell / nil
```

Description

Returns the effective physical cell binding for an occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_physCell	Name of the physical cell.
nil	The physical cell name was not returned.

Example

```
physCell=cphGetPhysicalCell(physConfigID "I0/N0")
```

cphGetPhysicalLib

```
cphGetPhysicalLib(
     g_physConfigID
     t_path
     )
     => t_physLib / nil
```

Description

Returns the effective physical library binding for an occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_physLib	Name of the physical library.
nil	The physical library name was not returned.

Example

```
physLib=cphGetPhysicalLib(physConfigID "I0/N0")
```

cphGetPhysicalView

```
cphGetPhysicalView(
     g_physConfigID
     t_path
     )
     => t_physView / nil
```

Description

Returns the effective physical view binding for an occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_physView	Name of the physical view.
nil	The physical view name was not returned.

Example

```
physView=cphGetPhysicalView(physConfigID "I0/N0")
```

cphGetStopPoint

```
cphGetStopPoint(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns whether or not the stop point attribute is set for a specific occurrence in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	Hierarchy traversal is set to stop at the specified occurrence.
nil	Hierarchy traversal is not set to stop at the specified occurrence.

Example

```
stopPoint=cphGetStopPoint(physConfigID "I0/N0")
```

cphGetViewBinding

```
cphGetViewBinding(
     g_physConfigID
     t_path
     )
     => t_logViewToUse / nil
```

Description

Returns the effective logical view to use for a specific occurrence of an instance in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_logViewToUse	Name of the logical view to use.
nil	The occurrence binding was not returned.

Example

```
logViewToUse=cphGetViewBinding(physConfigID "I0/N0")
```

cphSetCellForceDescend

```
cphSetCellForceDescend(
     g_physConfigID
     t_logLib
     t_logCell
     g_value
     )
     => t / nil
```

Description

Sets or unsets the force descend attribute on a specified cell in the physical configuration associated with the given ID. This forces hierarchy traversal to proceed beyond instances of the cell, even if it would normally be considered a leaf-level instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_value	Specifies whether force descend is set. ■ When set to `t`, the force descend is turned ON. ■ When set to `nil`, the force descend is turned OFF.

Value Returned

`t`	The force descend was successfully updated.
`nil`	The command failed.

Example

```
cphSetCellForceDescend(cph "cph" "nand" t)
```

cphSetCellPhysicalBinding

```
cphSetCellPhysicalBinding(
     g_physConfigID
     t_logLib
     t_logCell
     t_physLib
     t_physCell
     t_physView
     )
     => t / nil
```

Description

Sets the physical binding for a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_physLib	Name of the library containing the physical view.
t_physCell	Name of the cell containing the physical view.
t_physView	Name of the physical view.

Value Returned

t	The physical binding was set.
nil	The physical binding was not set.

Example

```
cphSetCellPhysicalBinding(physConfigID "cph" "nand" "cph" "nand" "layout")
```

cphSetCellStopList

```
cphSetCellStopList(
     g_physConfigID
     t_logLib
     t_logCell
     t_stopList
     )
     => t / nil
```

Description

Sets the physical stop view list for a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_stopList	List of physical view names used to determine the corresponding physical view for a given logical view. When traversing a hierarchy, Configure Physical Hierarchy stops when it encounters a view with one of the specified names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The physical stop view list was set.
nil	The physical stop view list was not set.

Example

```
cphSetCellStopList(physConfigID "cph" "nand" "layout abstract")
```

cphSetCellViewBinding

```
cphSetCellViewBinding(
     g_physConfigID
     t_logLib
     t_logCell
     t_logViewToUse
     )
     => t / nil
```

Description

Sets the logical view to use for a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_logViewToUse	Name of the logical view to use.

Value Returned

t	The cellview binding was set.
nil	The cellview binding was not set.

Example

```
cphSetCellViewBinding(physConfigID "cph" "nand" "sch")
```

cphSetCellViewList

```
cphSetCellViewList(
     g_physConfigID
     t_logLib
     t_logCell
     t_switchList
     )
     => t / nil
```

Description

Specifies the inherited logical switch view list for a specified logical cell in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_switchList	List of logical view names used to descend into a hierarchical design. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The inherited switch list was set.
nil	The inherited switch list was not set.

Example

```
cphSetCellViewList(physConfigID "cph" "nand" "physconfig sch symbol")
```

cphSetInstForceDescend

```
cphSetInstForceDescend(
     g_physConfigID
     t_lib
     t_cell
     t_view
     t_inst
     g_boolean
     )
     => t / nil
```

Description

Sets or unsets the force descend attribute on a specified instance in the physical configuration associated with the given ID. This forces hierarchy traversal to proceed beyond the instance, even if it would normally be considered a leaf instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_lib	Name of the library where concerned instance is instantiated.
t_cell	Name of the cell where concerned instance is instantiated.
t_view	Name of the cellview where concerned instance is instantiated.
t_inst	Name of the instance.
g_boolean	Boolean specifying whether the force descend is to be set (`t`) or unset (`nil`).

Value Returned

`t`	The force descend was set or unset on the specified instance.
`nil`	The force descend could not be set or unset.

Example

This example unsets the force descend attribute for an instance `I0`.

```
cphSetInstForceDescend(physConfigID "cph" "TopCell" "schematic" "I0" nil)
```

cphSetSchInstForceDescend

```
cphSetSchInstForceDescend(
     d_topSchCVID
     d_schInstID
     g_value
     g_all
     [ ?layCVId d_layCVId ]
     )
     => t / nil
```

Description

Sets the value for force descend on the specified instance, specifying top-level schematic and layout cellview IDs instead of the cphManager ID.

Arguments

d_topSchCVID

Database ID of the top-level schematic cellview.

d_schInstID

Database ID of the schematic instance.

g_value

If set to `t`, sets force descend on the specified schematic instance.

If set to `nil`, removes the force descend.

g_all

If set to `t`, updates the sub level physConfig views, if the specified instance ID is inside a sub level physConfig.

If set to `nil`, updates no sub level physConfigs.

?layCVId *d_layCVId*

Database ID of the top-level layout cellview.

Value Returned

`t`	The force descend was set or unset on the specified instance.
`nil`	The force descend could not be set or unset.

Examples

Finds the physical binding for the top-level schematic instance "`I0`".

```
scv -> top-level schematic cellview ID in the VLS XL session
lcv -> top-level layout cellview ID in the VLS XL session
i0 -> Instance ID for instance "I0" in the top-level schematic cellview
n0 -> Instance ID for instance "N0" in the lower-level nand2_A schematic cellview
cphGetSchInstPhysicalBinding(scv i0 ?layCVId lcv)
```

Sets force descend to descend past the physical stopping point and bind next level down at the `nmos`.

```
cphSetSchInstForceDescend(scv i0 t t ?layCVId lcv)
```

cphSetInstPhysicalBinding

```
cphSetInstPhysicalBinding(
     g_physConfigID
     t_lib
     t_cell
     t_view
     t_inst
     t_physLib
     t_physCell
     t_physView
     )
     => t / nil
```

Description

Sets the physical binding for a specified instance in the physical configuration associated with the given ID The library and cell are empty, the system issues a warning message.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_lib	Name of the library where concerned instance is instantiated.
t_cell	Name of the cell where concerned instance is instantiated.
t_view	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.
t_physLib	Name of the library containing the physical view.
t_physCell	Name of the cell containing the physical view.
t_physView	Name of the physical view.

Value Returned

t	The physical binding for the instance was set.
nil	The physical binding was not set.

Example

```
cphSetInstPhysicalBinding(physConfigID "cph" "TopCell" "schematic" "I0" "cph"
"inv" "layout")
```

cphSetSchCellPhysicalBinding

```
cphSetSchCellPhysicalBinding(
     d_topSchCVID
     t_libName
     t_cellName
     t_value
     [ ?layCVId d_layCVId ]
     )
     => t / nil
```

Description

Sets the physical binding of a cell specifying top-level schematic and layout cellviews IDs instead of the cphManager ID. After the cell binding is set, the SKILL function also saves the update to the physConfig view.

Arguments

d_topSchCVID

Database ID of the top-level schematic cellview.

t_libName

Name of the library containing the cell to be updated.

t_cellName

Name of the cell to be updated.

t_value

Physical library, cell, and view name represented in a string format.

?layCVId *d_layCVId*

Database ID of the top-level layout cellview associated with the specified cell.

Value Returned

`t`	The physical binding was set.
`nil`	The physical binding was not set.

Example

Sets cell binding for the cell "`nand`".

```
scv -> Schematic cellview ID of the top-level schematic in a VLS XL session
lcv -> Layout cellview ID of the top-level layout in a VLS XL session
cphSetSchCellPhysicalBinding(scv "cph" "nand" "cph nand layout2" ?layCVId lcv) => t
```

cphSetSchInstPhysicalBinding

```
cphSetSchInstPhysicalBinding(
     d_topSchCVID
     d_schInstID
     t_value
     g_all
     [ ?layCVId d_layCVId ]
     )
     => t / nil
```

Description

Sets the physical binding of an instance specifying the top-level schematic and layout cellview IDs instead of the cphManager ID.

Arguments

d_topSchCVID

Database ID of the top-level schematic cellview.

d_schInstID

Database ID of the schematic instance.

t_value

Physical library, cell, and view names represented in a string format.

g_all

Updates the sub physConfig view, if the specified schematic instance to update is inside a sub level physConfig.

?layCVId *d_layCVId*

Database ID of the top-level layout cellview.

Value Returned

t	The physical binding for the instance was set.
nil	The physical binding was not set.

Example

Sets the physical binding on instance "N0", which is instantiated in schematic cellview nand2_A using the Layout XL session represented by scv and lcv.

```
scv -> top-level schematic cellview ID in a VLS XL session
lcv -> top-level layout cellview ID in a VLS XL session
i0 -> Instance ID for instance "I0" in the top-level schematic cellview
n0 -> Instance ID for instance "N0" in the lower-level nand2_A schematic cellview
cphSetSchInstPhysicalBinding(scv n0 "cph nmos layout2" t ?layCVId lcv)=> t
```

cphSetInstStopList

```
cphSetInstStopList(
     g_physConfigID
     t_lib
     t_cell
     t_view
     t_inst
     t_stopList
     )
     => t / nil
```

Description

Sets an inherited physical stop view list for a specified instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_lib	Name of the library where concerned instance is instantiated.
t_cell	Name of the cell where concerned instance is instantiated.
t_view	Name of the cellview where concerned instance is instantiated.
t_inst	Name of the instance.
t_stopList	List of physical view names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The inherited stop list was set.
nil	The inherited stop list was not set.

Example

```
cphSetInstStopList(physConfigID "cph" "TopCell" "schematic" "I0" "layout
layoutX")
```

cphSetInstStopPoint

```
cphSetInstStopPoint(
     g_physConfigID
     t_lib
     t_cell
     t_view
     t_inst
     g_boolean
     )
     => t / nil
```

Description

Sets or unsets the stop point attribute on a specified instance in the physical configuration associated with the given ID. This means that hierarchy traversal always stops at this instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_lib	Name of the library where concerned instance is instantiated.
t_cell	Name of the cell where concerned instance is instantiated.
t_view	Name of the cellview where concerned instance is instantiated.
t_inst	Name of the instance.
g_boolean	Boolean specifying whether the stop point is to be set (`t`) or not (`nil`).

Value Returned

`t`	The stop point was set or unset on the specified instance.
`nil`	The stop point could not be set or unset.

Example

The example sets the stop point attribute for an instance `I0`.

```
cphSetInstStopPoint(physConfigID "cph" "TopCell" "schematic" "I0" t)
```

cphSetInstViewBinding

```
cphSetInstViewBinding(
     g_physConfigID
     t_lib
     t_cell
     t_view
     t_inst
     t_logViewToUse
     )
     => t / nil
```

Description

Sets the logical view to use for a specified instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_lib	Name of the library where concerned instance is instantiated.
t_cell	Name of the cell where concerned instance is instantiated.
t_view	Name of the cellview where concerned instance is instantiated.
t_inst	Name of the instance.
t_logViewToUse	Name of the logical view to use.

Value Returned

t	The instance binding was set.
nil	The instance binding was not set.

Example

```
cphSetInstViewBinding(physConfigID "cph" "TopCell" "schematic" "I0" "schematic")
```

cphSetInstViewList

```
cphSetInstViewList(
     g_physConfigID
     t_lib
     t_cell
     t_view
     t_inst
     t_switchList
     )
     => t / nil
```

Description

Sets an inherited logical switch view list for a specified instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_lib	Name of the library where concerned instance is instantiated.
t_cell	Name of the cell where concerned instance is instantiated.
t_view	Name of the cellview where concerned instance is instantiated.
t_inst	Name of the instance.
t_switchList	List of logical view names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The inherited switch list was set.
nil	The inherited switch list was not set.

Example

```
cphSetInstViewList(physConfigID "cph" "TopCell" "schematic" "I0" "schematic symbo
l")
```

cphSetOccurForceDescend

```
cphSetOccurForceDescend(
     g_physConfigID
     t_path
     g_boolean
     )
     => t / nil
```

Description

Sets or unsets the force descend attribute on a specific occurrence of an instance in the physical configuration associated with the given ID. This forces hierarchy traversal to proceed beyond the occurrence, even if it would normally be considered a leaf instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_boolean	Boolean specifying whether the force descend is set (`t`) or unset (`nil`).

Value Returned

`t`	The force descend was set on the specified occurrence.
`nil`	The force descend was not set.

Example

This example unsets the force descend attribute for an occurrence `I0/N0`.

```
cphSetOccurForceDescend(physConfigID "I0/N0" nil)
```

cphSetOccurPhysicalBinding

```
cphSetOccurPhysicalBinding(
     g_physConfigID
     t_path
     t_physLib
     t_physCell
     t_physView
     )
     => t / nil
```

Description

Sets the physical binding for a specific occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_physLib	Name of the library containing the physical view.
t_physCell	Name of the cell containing the physical view.
t_physView	Name of the physical view.

Value Returned

t	The physical binding for the occurrence was set.
nil	The physical binding was not set.

Example

```
cphSetOccurPhysicalBinding(physConfigID "I0/N0" "cph" "nmos" "layout")
```

cphSetOccurStopList

```
cphSetOccurStopList(
     g_physConfigID
     t_path
     t_stopList
     )
     => t / nil
```

Description

Sets an inherited physical stop view list for a specific occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_stopList	List of physical view names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The inherited stop list was set for the specified occurrence.
nil	The inherited stop list was not set for the specified occurrence

Example

```
cphSetOccurStopList(physConfigID "I0/N0" "layout layoutY")
```

cphSetOccurStopPoint

```
cphSetOccurStopPoint(
     g_physConfigID
     t_path
     g_boolean
     )
     => t / nil
```

Description

Sets or unsets the stop point attribute on a specified occurrence of an instance in the physical configuration associated with the given ID. This means that hierarchy traversal always stops at this instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_boolean	Boolean specifying whether the stop point is to be set (`t`) or unset (`nil`).

Value Returned

`t`	The stop point was set or unset on the specified occurrence.
`nil`	The stop point could not be set.

Example

This example unsets the stop point attribute for an occurrence `I0/N0`.

```
cphSetOccurStopPoint(physConfigID "I0/N0" nil)
```

cphSetOccurViewBinding

```
cphSetOccurViewBinding(
     g_physConfigID
     t_path
     t_logViewToUse
     )
     => t / nil
```

Description

Sets the logical view to use for a specific occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_logViewToUse	Name of the logical view to use.

Value Returned

t	The occurrence binding was set.
nil	The occurrence binding was not set.

Example

```
cphSetOccurViewBinding(physConfigID "I0/N0" "sch2")
```

cphSetOccurViewList

```
cphSetOccurViewList(
     g_physConfigID
     t_path
     t_switchList
     )
     => t / nil
```

Description

Sets an inherited logical switch view list for a specific occurrence of an instance in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_switchList	List of logical view names. Separate each name with a space and enclose the list in quotation marks.

Value Returned

t	The inherited switch list was set for the specified occurrence.
nil	The inherited switch list was not set for the specified occurrence

Example

```
cphSetOccurViewList(physConfigID "I0/N0" "physconfig sch")
```

Hierarchy Configuration Attributes Pane Functions

Use the functions described in this section to set and retrieve the attribute values associated with cells, instances, and occurrences in the hierarchy configuration table views.

For more information on these tables see, Hierarchy Configuration Mode in the CPH Window.

cphDeleteCellFingerSplit

cphDeleteCellIgnoreForCheck

cphDeleteCellIgnoreForGen

cphDeleteCellMFactorSplit

cphDeleteCellParamIgnoreForCheck

cphDeleteCellParamIgnoreForGen

cphDeleteCellParamNameMapping

cphDeleteCellParamToCheck

cphDeleteCellRemoveDevice

cphDeleteCellRounding

cphDeleteCellSFactorSplit

cphDeleteCellTermIgnoreForCheck

cphDeleteCellTermIgnoreForGen

cphDeleteCellTermNameMapping

cphDeleteCellVLGen

cphDeleteInstFingerSplit

cphDeleteInstIgnoreForCheck

cphDeleteInstIgnoreForGen

cphDeleteInstMFactorSplit

cphDeleteInstParamIgnoreForCheck

cphDeleteCellFingerSplit

```
cphDeleteCellFingerSplit(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Split fingered devices* attribute defined for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Split fingered devices* attribute was deleted.
nil	The *Split fingered devices* attribute was not deleted.

Example

```
cphDeleteCellFingerSplit(physConfigID "cph" "nand")
```

cphDeleteCellIgnoreForCheck

```
cphDeleteCellIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Ignore for check* attribute defined for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Ignore for check* attribute was deleted.
nil	The *Ignore for check* attribute was not deleted.

Example

```
cphDeleteCellIgnoreForCheck(physConfigID "cph" "nand")
```

cphDeleteCellIgnoreForGen

```
cphDeleteCellIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Ignore for layout generation* attribute defined for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Ignore for layout generation* attribute was deleted.
nil	The *Ignore for layout generation* attribute was not deleted.

Example

```
cphDeleteCellIgnoreForGen(physConfigID "cph" "nand")
```

cphDeleteCellMFactorSplit

```
cphDeleteCellMFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Split mfactored devices* attribute defined for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Split mfactored devices* attribute was deleted.
nil	The *Split mfactored devices* attribute was not deleted.

Example

```
cphDeleteCellMFactorSplit(physConfigID "cph" "nand")
```

cphDeleteCellParamIgnoreForCheck

```
cphDeleteCellParamIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the list of parameter names to be ignored when running *Check Against Source* on the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The list of parameter names was deleted.
nil	The list of parameter names was not deleted.

Example

```
cphDeleteCellParamIgnoreForCheck(physConfigID "cph" "nand")
```

cphDeleteCellParamIgnoreForGen

```
cphDeleteCellParamIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the list of parameter names to be ignored when generating a layout for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The list of parameter names was deleted.
nil	The list of parameter names was not deleted.

Example

```
cphDeleteCellParamIgnoreForGen(physConfigID "cph" "nand")
```

cphDeleteCellParamNameMapping

```
cphDeleteCellParamNameMapping(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the mapping specified between parameter names in the schematic and layout views of the specified cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the logical cell.

Value Returned

t	Parameter name mapping was deleted.
nil	Parameter name mapping was not deleted.

Example

```
cphDeleteCellParamNameMapping(physConfigID "cph" "nand")
```

cphDeleteCellParamToCheck

```
cphDeleteCellParamToCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the list of parameter names to be checked when running *Check Against Source* for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The list of parameter names to be checked was deleted.
nil	The list of parameter names to be checked was not deleted.

Example

```
cphDeleteCellParamToCheck(physConfigID "cph" "nand")
```

cphDeleteCellRemoveDevice

```
cphDeleteCellRemoveDevice(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Remove device* rule defined for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Remove device* rule was deleted.
nil	The *Remove device* rule was not deleted.

Example

```
cphDeleteCellRemoveDevice(physConfigID "cph" "nand")
```

cphDeleteCellRounding

```
cphDeleteCellRounding(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Rounding* rule defined for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Rounding* rule was deleted.
nil	The *Rounding* rule was not deleted.

Example

```
cphDeleteCellRounding(physConfigID "cph" "nand")
```

cphDeleteCellSFactorSplit

```
cphDeleteCellSFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the *Split sfactored devices* attribute defined for the specified logical cell in the physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Split sfactored devices* attribute was deleted.
nil	The *Split sfactored devices* attribute was not deleted.

Example

```
cphDeleteCellSFactorSplit(physConfigID "cph" "nand")
```

cphDeleteCellTermIgnoreForCheck

```
cphDeleteCellTermIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the list of terminal names to be ignored when running *Check Against Source* on the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The list of terminal names was deleted.
nil	The list of terminal names was not deleted.

Example

```
cphDeleteCellTermIgnoreForCheck(physConfigID "cph" "nand")
```

cphDeleteCellTermIgnoreForGen

```
cphDeleteCellTermIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the list of terminal names to be ignored when generating a layout for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The list of terminal names was deleted.
nil	The list of terminal names was not deleted.

Example

```
cphDeleteCellTermIgnoreForGen(physConfigID "cph" "nand")
```

cphDeleteCellTermNameMapping

```
cphDeleteCellTermNameMapping(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the mapping specified between terminal names in the schematic and layout views of the specified cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the logical cell.

Value Returned

t	Terminal name mapping was deleted.
nil	Terminal name mapping was not deleted.

Example

```
val=cphDeleteCellTermNameMapping(physConfigID "cph" "nand")
```

cphDeleteCellVLGen

```
cphDeleteCellVLGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the VLGen property and physical binding of the specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The VLGen property was deleted.
nil	The VLGen property was not deleted.

Example

```
cphDeleteCellVLGen(physConfigID "cph" "nand2")
```

cphDeleteInstFingerSplit

```
cphDeleteInstFingerSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     )
     => t / nil
```

Description

Deletes the *Split fingered devices* attribute defined for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.

Value Returned

t	The *Split fingered devices* attribute was deleted.
nil	The *Split fingered devices* attribute was not deleted.

Example

```
cphDeleteInstFingerSplit(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstIgnoreForCheck

```
cphDeleteInstIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the *Ignore for check* attribute defined for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

`t`	The *Ignore for check* attribute was deleted.
`nil`	The *Ignore for check* attribute was not deleted.

Example

```
cphDeleteInstIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstIgnoreForGen

```
cphDeleteInstIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the *Ignore for layout generation* attribute defined for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The *Ignore for layout generation* attribute was deleted.
nil	The *Ignore for layout generation* attribute was not deleted.

Example

```
cphDeleteInstIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstMFactorSplit

```
cphDeleteInstMFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the *Split mfactored devices* attribute defined for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The *Split mfactored devices* attribute was deleted.
nil	The *Split mfactored devices* attribute was not deleted.

Example

```
cphDeleteInstMFactorSplit(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstParamIgnoreForCheck

```
cphDeleteInstParamIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the list of parameter names to be ignored when running *Check Against Source* on the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The list of parameter names was deleted.
nil	The list of parameter names was not deleted.

Example

```
cphDeleteInstParamIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstParamIgnoreForGen

```
cphDeleteInstParamIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the list of parameter names to be ignored when generating a layout for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The list of parameter names was deleted.
nil	The list of parameter names was not deleted.

Example

```
cphDeleteInstParamIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstParamToCheck

```
cphDeleteInstParamToCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the list of parameter names to be checked when running *Check Against Source* on the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The list of parameters to be checked was deleted.
nil	The list of parameters to be checked was not deleted.

Example

```
val=cphDeleteInstParamToCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstRemoveDevice

```
cphDeleteInstRemoveDevice(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the *Remove device* rule defined for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The *Remove device* rule was deleted.
nil	The *Remove device* rule was not deleted.

Example

```
cphDeleteInstRemoveDevice(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstRounding

```
cphDeleteInstRounding(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the *Rounding* rule defined for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The *Rounding device* rule was deleted.
nil	The *Rounding* rule was not deleted.

Example

```
cphDeleteInstRounding(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstSFactorSplit

```
cphDeleteInstSFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the *Split sfactored devices* attribute defined for the specified instance in the physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The *Split sfactored devices* attribute was deleted.
nil	The *Split sfactored devices* attribute was not deleted.

Example

```
cphDeleteInstSFactorSplit(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstTermIgnoreForCheck

```
cphDeleteInstTermIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the list of terminal names to be ignored when running *Check Against Source* on the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The list of terminal names was deleted.
nil	The list of terminal names was not deleted.

Example

```
cphDeleteInstTermIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteInstTermIgnoreForGen

```
cphDeleteInstTermIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Deletes the list of terminal names to be ignored when generating a layout for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The list of terminal names was deleted.
nil	The list of terminal names was not deleted.

Example

```
cphDeleteInstTermIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphDeleteOccurFingerSplit

```
cphDeleteOccurFingerSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Split fingered devices* attribute defined for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The *Split fingered devices* attribute was deleted.
nil	The *Split fingered devices* attribute was not deleted.

Example

```
cphDeleteOccurFingerSplit(physConfigID "I0")
```

cphDeleteOccurIgnoreForCheck

```
cphDeleteOccurIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Ignore for check* attribute defined for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The *Ignore for check* attribute was deleted.
nil	The *Ignore for check* attribute was not deleted.

Example

```
cphDeleteOccurIgnoreForCheck(physConfigID "I0")
```

cphDeleteOccurIgnoreForGen

```
cphDeleteOccurIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Ignore for layout generation* attribute defined for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The *Ignore for layout generation* attribute was deleted.
`nil`	The *Ignore for layout generation* attribute was not deleted.

Example

```
cphDeleteOccurIgnoreForGen(physConfigID "I0")
```

cphDeleteOccurMFactorSplit

```
cphDeleteOccurMFactorSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Split mfactored devices* attribute defined for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The *Split mfactored devices* attribute was deleted.
nil	The *Split mfactored devices* attribute was not deleted.

Example

```
cphDeleteOccurMFactorSplit(physConfigID "I0")
```

cphDeleteOccurParamIgnoreForCheck

```
cphDeleteOccurParamIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the list of parameter names to be ignored when running *Check Against Source* on the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The list of parameter names was deleted.
nil	The list of parameter names was not deleted.

Example

```
cphDeleteOccurParamIgnoreForCheck(physConfigID "I0/N0")
```

cphDeleteOccurParamIgnoreForGen

```
cphDeleteOccurParamIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the list of parameter names to be ignored when generating a layout for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The list of parameter names was deleted.
`nil`	The list of parameter names was not deleted.

Example

```
cphDeleteOccurParamIgnoreForGen(physConfigID "I0/N0")
```

cphDeleteOccurParamToCheck

```
cphDeleteOccurParamToCheck(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the list of parameter names to be checked when running *Check Against Source* for the specific occurrence of an instance in the physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The list of parameter names to be checked was deleted.
nil	The list of parameter names to be checked was not deleted.

Example

```
val=cphDeleteOccurParamToCheck(physConfigID "I0/N0")
```

cphDeleteOccurRemoveDevice

```
cphDeleteOccurRemoveDevice(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Remove device* rule defined for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The *Remove device* rule was deleted.
nil	The *Remove device* rule was not deleted.

Example

```
cphDeleteOccurRemoveDevice(physConfigID "I0")
```

cphDeleteOccurRounding

```
cphDeleteOccurRounding(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Rounding* rule defined for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The *Rounding* rule was deleted.
`nil`	The *Rounding* rule was not deleted.

Example

```
cphDeleteOccurRounding(physConfigID "I0/N0")
```

cphDeleteOccurSFactorSplit

```
cphDeleteOccurSFactorSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the *Split sfactored devices* attribute defined for the specific occurrence of an instance in the physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The *Split sfactored devices* attribute was deleted.
nil	The *Split sfactored devices* attribute was not deleted.

Example

```
cphDeleteOccurSFactorSplit(physConfigID "I0/N0")
```

cphDeleteOccurTermIgnoreForCheck

```
cphDeleteOccurTermIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the list of terminal names to be ignored when running *Check Against Source* on the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The list of terminal names was deleted.
nil	The list of terminal names was not deleted.

Example

```
cphDeleteOccurTermIgnoreForCheck(physConfigID "I0/N0")
```

cphDeleteOccurTermIgnoreForGen

```
cphDeleteOccurTermIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Deletes the list of terminal names to be ignored when generating a layout for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The list of terminal names was deleted.
`nil`	The list of terminal names was not deleted.

Example

```
cphDeleteOccurTermIgnoreForGen(physConfigID "I0/N0")
```

cphGetCellFingerSplit

```
cphGetCellFingerSplit(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the value of the *Split fingered devices* attribute for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Split fingered devices* attribute is selected for the specified cell.
nil	The *Split fingered devices* attribute is deselected for the specified cell.

Example

```
val=cphGetCellFingerSplit(physConfigID "cph" "nand")
```

cphGetCellIgnoreForCheck

```
cphGetCellIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the value of the *Ignore for check* attribute for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The specified cell will be ignored for check.
nil	The specified cell will not be ignored for check.

Example

```
val=cphGetCellIgnoreForCheck(physConfigID "cph" "nand")
```

cphGetCellIgnoreForGen

```
cphGetCellIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the value of the *Ignore for layout generation* attribute for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The specified cell will be ignored.
nil	The specified cell will not be ignored.

Example

```
val=cphGetCellIgnoreForGen(physConfigID "cph" "nand")
```

cphGetCellMFactorSplit

```
cphGetCellMFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the value of the *Split mfactored devices* attribute for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	*Split mfactored devices* is switched on for the specified cell.
nil	*Split mfactored devices* is switched off for the specified cell.

Example

```
val=cphGetCellMFactorSplit(physConfigID "cph" "nand")
```

cphGetCellParamIgnoreForCheck

```
cphGetCellParamIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be ignored when running *Check Against Source* on the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for check.
nil	The value could not be returned.

Example

```
val=cphGetCellParamIgnoreForCheck(physConfigID "cph" "nand")
```

cphGetCellParamIgnoreForGen

```
cphGetCellParamIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be ignored when generating a layout for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for generation.
nil	The value could not be returned.

Example

```
val=cphGetCellParamIgnoreForGen(physConfigID "cph" "nand")
```

cphGetCellParamNameMapping

```
cphGetCellParamNameMapping(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_paramMapping / nil
```

Description

Returns the mapping specified between parameter names in the schematic and layout views of the specified cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.

Value Returned

t_paramMapping	String specifying how parameter names are mapped.
nil	Parameter name mapping was not set.

Example

```
val=cphGetCellParamNameMapping(physConfigID "cph" "nand")
```

cphGetCellParamToCheck

```
cphGetCellParamToCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be checked when running *Check Against Source* on the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_listOfParamNames	List of parameter names to be checked.
nil	The value could not be returned.

Example

```
val=cphGetCellParamToCheck(physConfigID "cph" "nand")
```

cphGetCellRemoveDevice

```
cphGetCellRemoveDevice(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_removeDevice / nil
```

Description

Returns the value of the *Remove device* rule for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_removeDevice	Value of the *Remove device* rule.
nil	The value could not be returned.

Example

```
val=cphGetCellRemoveDevice(physConfigID "cph" "nand")
```

cphGetCellRounding

```
cphGetCellRounding(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_roundingRule / nil
```

Description

Returns the value of the *Rounding* rule for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_roundingRule	Value of the *Rounding* rule.
nil	The value could not be returned.

Example

```
val=cphGetCellRounding(physConfigID "cph" "nand")
```

cphGetCellSFactorSplit

```
cphGetCellSFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the value of the *Split sfactored devices* attribute for a specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The *Split sfactored devices* attribute is set for the specified cell.
nil	The *Split sfactored devices* attribute is not set for the specified cell.

Example

```
val=cphGetCellSFactorSplit(physConfigID "cph" "nand")
```

cphGetCellTermIgnoreForCheck

```
cphGetCellTermIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_listOfTermNames / nil
```

Description

Returns the list of terminals to be ignored when running *Check Against Source* for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for check.
nil	The list of terminals to be ignored was not set.

Example

```
val=cphGetCellTermIgnoreForCheck(physConfigID "cph" "nand")
```

cphGetCellTermIgnoreForGen

```
cphGetCellTermIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_listOfTermNames / nil
```

Description

Returns the list of terminals to be ignored when generating a layout for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for generation.
nil	List of terminals to be ignored could not be returned.

Example

```
val=cphGetCellTermIgnoreForGen(physConfigID "cph" "nand")
```

cphGetCellTermNameMapping

```
cphGetCellTermNameMapping(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_termMapping / nil
```

Description

Returns the mapping between terminal names in the schematic and layout views of the specified cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.

Value Returned

t_termMapping	String specifying how terminal names are mapped.
nil	The value could not be returned.

Example

```
val=cphGetCellTermNameMapping(physConfigID "cph" "nand")
```

cphGetCellVLGen

```
cphGetCellVLGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the VLGen property mapped to a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The VLGen property is set on the specified cell.
nil	The VLGen property is not set on the specified cell.

Example

```
cphGetCellVLGen(physConfigID "cph" "nand2")
```

cphGetFingerSplit

```
cphGetFingerSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the effective value of the *Split fingered devices* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The *Split fingered devices* attribute is selected for the specified occurrence.
`nil`	The *Split fingered devices* attribute is deselected for the specified occurrence.

Example

```
val=cphGetFingerSplit(physConfigID "I0/N0")
```

cphGetIgnoreForCheck

```
cphGetIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the effective value of the *Ignore for check* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The specified occurrence will be ignored for check.
nil	The specified occurrence will not be ignored for check.

Example

```
val=cphGetIgnoreForCheck(physConfigID "I0/N0")
```

cphGetIgnoreForGen

```
cphGetIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the effective value of the *Ignore for layout generation* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The specified occurrence will be ignored.
`nil`	The specified occurrence will not be ignored.

Example

```
val=cphGetIgnoreForGen(physConfigID "I0/N0")
```

cphGetInstFingerSplit

```
cphGetInstFingerSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Returns the value of the *Split fingered devices* attribute for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t	The *Split fingered devices* attribute is selected for the specified instance.
nil	The *Split fingered devices* attribute is deselected for the specified instance.

Example

```
val=cphGetInstFingerSplit(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstIgnoreForCheck

```
cphGetInstIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Returns the value of the *Ignore for check* attribute for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t	The specified instance will be ignored for check.
nil	The specified instance will not be ignored for check.

Example

```
val=cphGetInstIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstIgnoreForGen

```
cphGetInstIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Returns the value of the *Ignore for layout generation* attribute for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t	The specified instance will be ignored.
nil	The specified instance will not be ignored.

Example

```
val=cphGetInstIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstMFactorSplit

```
cphGetInstMFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Returns the value of the *Split mfactored devices* attribute for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

`t`	*Split mfactored devices* is switched on for the specified instance.
`nil`	*Split mfactored devices* is switched off for the specified instance.

Example

```
val=cphGetInstMFactorSplit(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstParamIgnoreForCheck

```
cphGetInstParamIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be ignored when running *Check Against Source* on the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for check.
nil	The value could not be returned.

Example

```
val=cphGetInstParamIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstParamIgnoreForGen

```
cphGetInstParamIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be ignored when generating a layout for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for generation.
nil	The value could not be returned.

Example

```
val=cphGetInstParamIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstParamToCheck

```
cphGetInstParamToCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be checked when running *Check Against Source* on the specified instances in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t_listOfParamNames	List of parameter names to be checked.
nil	The value could not be returned.

Example

```
val=cphGetInstParamToCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstRemoveDevice

```
cphGetInstRemoveDevice(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_removeDevice / nil
```

Description

Returns the value of the *Remove device* rule for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_removeDevice	Value of the *Remove device* rule.
nil	The value could not be returned.

Example

```
val=cphGetInstRemoveDevice(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstRounding

```
cphGetInstRounding(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_roundingRule / nil
```

Description

Returns the value of the *Rounding* rule for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_roundingRule	Value of the *Rounding* rule.
nil	The value could not be returned.

Example

```
val=cphGetInstRounding(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstSFactorSplit

```
cphGetInstSFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t / nil
```

Description

Returns the value of the *Split sfactored devices* attribute for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.

Value Returned

t	The *Split sfactored devices* attribute is set for the specified instance.
nil	The *Split sfactored devices* attribute is not set for the specified instance.

Example

```
val=cphGetInstSFactorSplit(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstTermIgnoreForCheck

```
cphGetInstTermIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_listOfTermNames / nil
```

Description

Returns the list of terminals to be ignored by when running *Check Against Source* for a specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for check.
nil	List of terminals to be ignored was not set.

Example

```
val=cphGetInstTermIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetInstTermIgnoreForGen

```
cphGetInstTermIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     )
     => t_listOfTermNames / nil
```

Description

Returns the list of terminals to be ignored when generating a layout for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the view where concerned instance is instantiated.
t_inst	Name of the instance.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for generation.
nil	The value could not be returned.

Example

```
val=cphGetInstTermIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0")
```

cphGetMFactorSplit

```
cphGetMFactorSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the effective value of the *Split mfactored devices* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	*Split mfactored devices* is switched on for the specified occurrence.
`nil`	*Split mfactored devices* is switched off for the specified occurrence.

Example

```
val=cphGetMFactorSplit(physConfigID "I0/N0")
```

cphGetOccurFingerSplit

```
cphGetOccurFingerSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the value of the *Split fingered devices* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The *Split fingered devices* attribute is selected for the specified occurrence.
`nil`	The *Split fingered devices* attribute is deselected for the specified occurrence.

Example

```
val=cphGetOccurFingerSplit(physConfigID "I0/N0")
```

cphGetOccurIgnoreForCheck

```
cphGetOccurIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the value of the *Ignore for check* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t	The specified occurrence will be ignored for check.
nil	The specified occurrence will not be ignored for check.

Example

```
val=cphGetOccurIgnoreForCheck(physConfigID "I0/N0")
```

cphGetOccurIgnoreForGen

```
cphGetOccurIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the value of the *Ignore for layout generation* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The specified occurrence will be ignored.
`nil`	The specified occurrence will not be ignored.

Example

```
val=cphGetOccurIgnoreForGen(physConfigID "I0/N0")
```

cphGetOccurMFactorSplit

```
cphGetOccurMFactorSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the value of the *Split mfactored devices* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	*Split mfactored devices* is switched on for the specified occurrence.
`nil`	*Split mfactored devices* is switched off for the specified occurrence.

Example

```
val=cphGetOccurMfactorSplit(physConfigID "I0/N0")
```

cphGetOccurParamIgnoreForCheck

```
cphGetOccurParamIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be ignored when running *Check Against Source* on the specified view in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for check.
nil	There is no list of parameter names set for the occurrence.

Example

```
val=cphGetOccurParamIgnoreForCheck(physConfigID "I0/N0")
```

cphGetOccurParamIgnoreForGen

```
cphGetOccurParamIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be ignored when generating a layout for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for generation.
nil	There is no list of parameter names set for the occurrence.

Example

```
val=cphGetOccurParamIgnoreForGen(physConfigID "I0/N0")
```

cphGetOccurParamToCheck

```
cphGetOccurParamToCheck(
     g_physConfigID
     t_path
     )
     => t_listOfParamNames / nil
```

Description

Returns the list of parameter names to be checked when running *Check Against Source* for the specific occurrence of an instance in the physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfParamNames	List of parameter names to be checked.
nil	There is no list of parameter names set for the occurrence.

Example

```
val=cphGetOccurParamToCheck(physConfigID "I0/N0")
```

cphGetOccurRemoveDevice

```
cphGetOccurRemoveDevice(
     g_physConfigID
     t_path
     )
     => t_removeDevice / nil
```

Description

Returns the *Remove device* rule for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_removeDevice	Value of the *Remove device* rule.
nil	There is no *Remove device* rule set for the occurrence.

Example

```
val=cphGetOccurRemoveDevice(physConfigID "I0/N0")
```

cphGetOccurRounding

```
cphGetOccurRounding(
     g_physConfigID
     t_path
     )
     => t_roundingRule / nil
```

Description

Returns the *Rounding* rule for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_roundingRule	Value of the *Rounding* rule.
nil	There is no *Rounding* rule set for the occurrence.

Example

```
val=cphGetOccurRounding(physConfigID "I0/N0")
```

cphGetOccurSFactorSplit

```
cphGetOccurSFactorSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the value of the *Split sfactored devices* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The *Split sfactored devices* attribute is set for the specified occurrence.
`nil`	The *Split sfactored devices* attribute is not set for the specified occurrence.

Example

```
val=cphGetOccurSFactorSplit(physConfigID "I0/N0")
```

cphGetOccurTermIgnoreForCheck

```
cphGetOccurTermIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t_listOfTermNames / nil
```

Description

Returns the list of terminals to be ignored by when running *Check Against Source* for a specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for check.
nil	There is no list of terminal names set for the occurrence.

Example

```
val=cphGetOccurTermIgnoreForCheck(physConfigID "I0/N0")
```

cphGetOccurTermIgnoreForGen

```
cphGetOccurTermIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t_listOfTermNames / nil
```

Description

Returns the list of terminals to be ignored when generating a layout for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for generation.
nil	There is no list of terminal names set for the occurrence.

Example

```
val=cphGetOccurTermIgnoreForGen(physConfigID "I0/N0")
```

cphGetParamIgnoreForCheck

```
cphGetParamIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t_listOfParamNames / nil
```

Description

Returns the effective list of parameter names to be ignored by *Check Against Source* for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for check.
nil	The value could not be returned.

Example

```
val=cphGetParamIgnoreForCheck(physConfigID "I0/N0")
```

cphGetParamIgnoreForGen

```
cphGetParamIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t_listOfParamNames / nil
```

Description

Returns the effective list of parameter names to be ignored when generating a layout for the specified occurrence of an instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfParamNames	List of parameter names to be ignored for generation.
nil	The value could not be returned.

Example

```
val=cphGetParamIgnoreForGen(physConfigID "I0/N0")
```

cphGetParamNameMapping

```
cphGetParamNameMapping(
     g_physConfigID
     t_path
     )
     => t_paramMapping / nil
```

Description

Returns the effective mappings between parameter names in the schematic and layout views of the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_paramMapping	String specifying how parameter names are mapped.
nil	Parameter name mapping was not set.

Example

```
val=cphGetParamNameMapping(physConfigID "I0/N0")
```

cphGetParamToCheck

```
cphGetParamToCheck(
     g_physConfigID
     t_path
     )
     => t_listOfParamNames / nil
```

Description

Returns the effective list of parameter names to be checked by *Check Against Source* for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfParamNames	List of parameter names to be checked.
nil	The value could not be returned.

Example

```
val=cphGetParamToCheck(physConfigID "I0/N0")
```

cphGetRemoveDevice

```
cphGetRemoveDevice(
     g_physConfigID
     t_path
     )
     => t_removeDevice / nil
```

Description

Returns the effective value of the *Remove device* rule for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_removeDevice	Value of the *Remove device* rule.
nil	The value could not be returned.

Example

```
val=cphGetRemoveDevice(physConfigID "I0/N0")
```

cphGetRounding

```
cphGetRounding(
     g_physConfigID
     t_path
     )
     => t_roundingRule / nil
```

Description

Returns the effective value of the *Rounding* rule for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_roundingRule	Value of the *Rounding* rule.
nil	The value could not be returned.

Example

```
val=cphGetRounding(physConfigID "I0/N0")
```

cphGetSFactorSplit

```
cphGetSFactorSplit(
     g_physConfigID
     t_path
     )
     => t / nil
```

Description

Returns the effective value of the *Split sfactored devices* attribute for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

`t`	The *Split sfactored devices* attribute is set for the specified occurrence.
`nil`	The *Split sfactored devices* attribute is not set for the specified occurrence.

Example

```
val=cphGetSFactorSplit(physConfigID "I0/N0")
```

cphGetTermIgnoreForCheck

```
cphGetTermIgnoreForCheck(
     g_physConfigID
     t_path
     )
     => t_listOfTermNames / nil
```

Description

Returns the effective list of terminals to be ignored when running *Check Against Source* for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for check.
nil	The value could not be returned.

Example

```
val=cphGetTermIgnoreForCheck(physConfigID "I0/N0")
```

cphGetTermIgnoreForGen

```
cphGetTermIgnoreForGen(
     g_physConfigID
     t_path
     )
     => t_listOfTermNames / nil
```

Description

Returns the effective list of terminals to be ignored when generating a layout for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_listOfTermNames	List of terminal names to be ignored for generation.
nil	The value could not be returned.

Example

```
val=cphGetTermIgnoreForGen(physConfigID "I0/N0")
```

cphGetTermNameMapping

```
cphGetTermNameMapping(
     g_physConfigID
     t_path
     )
     => t_termMapping / nil
```

Description

Returns the effective mappings between terminal names in the schematic and layout views of the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

Value Returned

t_termMapping	String specifying how terminal names are mapped.
nil	The value could not be returned.

Example

```
val=cphGetTermNameMapping(physConfigID "I0/N0")
```

cphIsParamIgnoredForCheck

```
cphIsParamIgnoredForCheck(
     g_physConfigID
     t_path
     t_paramName
     )
     => t / nil
```

Description

Returns whether the specified parameter on the specified occurrence is ignored for check.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_paramName	Name of the parameter.

Value Returned

t	The specified parameter is ignored for check.
nil	The specified parameter is not ignored for check.

Example

```
cphIsParamIgnoredForCheck(physConfigID "I0/N0" "m")
```

cphIsParamIgnoredForGen

```
cphIsParamIgnoredForGen(
     g_physConfigID
     t_path
     t_paramName
     )
     => t / nil
```

Description

Returns whether the specified parameter on the specified occurrence is ignored for generation.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_paramName	Name of the parameter.

Value Returned

t	The specified parameter is ignored for generation.
nil	The specified parameter is not ignored for generation.

Example

```
cphIsParamIgnoredForGen(physConfigID "I0/N0" "m")
```

cphIsTermIgnoredForCheck

```
cphIsTermIgnoredForCheck(
     g_physConfigID
     t_path
     t_logTermName
     )
     => t / nil
```

Description

Returns whether the specified logical terminal on the specified occurrence is ignored for check.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_logTermName	Name of the logical terminal.

Value Returned

t	The specified terminal is ignored for check.
nil	The specified terminal is not ignored for check.

Example

```
cphIsTermIgnoredForCheck(physConfigID "I0/N0" "S")
```

cphIsTermIgnoredForGen

```
cphIsTermIgnoredForGen(
     g_physConfigID
     t_path
     t_logTermName
     )
     => t / nil
```

Description

Returns whether the specified logical terminal on the specified occurrence is ignored for generation.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_logTermName	Name of the logical terminal.

Value Returned

t	The specified terminal is ignored for generation.
nil	The specified terminal is not ignored for generation.

Example

```
cphIsTermIgnoredForGen(physConfigID "I0/N0" "S")
```

cphSetCellFingerSplit

```
cphSetCellFingerSplit(
     g_physConfigID
     t_logLib
     t_logCell
     g_boolean
     )
     => t / nil
```

Description

Sets the *Split fingered devices* option for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_boolean	Boolean specifying the value of the *Split fingered devices* option. Valid Values: `t` or `nil`.

Value Returned

`t`	*Split fingered devices* attribute was set.
`nil`	*Split fingered devices* attribute was not set.

Example

```
cphSetCellFingerSplit(physConfigID "cph" "nand" t)
```

cphSetCellIgnoreForCheck

```
cphSetCellIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     g_boolean
     )
     => t / nil
```

Description

Sets the *Ignore for check* option for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_boolean	Boolean specifying the value of the *Ignore for check* option. Valid Values: `t` or `nil`.

Value Returned

`t`	*Ignore for check* was set.
`nil`	*Ignore for check* was not set.

Example

```
cphSetCellIgnoreForCheck(physConfigID "cph" "nand" nil)
```

cphSetCellIgnoreForGen

```
cphSetCellIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     g_value
     )
     => t / nil
```

Description

Sets the *Ignore for layout generation* option to `t` or `nil` for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the logical cell where concerned instance is instantiated.
g_boolean	Boolean specifying the value of the *Ignore for layout generation* option. Valid Values: `t` or `nil`.

Value Returned

`t`	*Ignore for layout generation* was set.
`nil`	*Ignore for layout generation* was not set.

Example

```
cphSetCellIgnoreForGen(physConfigID "cph" "nand" t)
```

cphSetCellMFactorSplit

```
cphSetCellMFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     g_boolean
     )
     => t / nil
```

Description

Sets the *Split mfactored devices* option to `t` or `nil` for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_boolean	Boolean specifying the value of the *Split mfactored devices* option. Valid Values: `t` or `nil`.

Value Returned

`t`	*Split mfactored devices* was set.
`nil`	*Split mfactored devices* was not set.

Example

```
cphSetCellMFactorSplit(physConfigID "cph" "nand" t)
```

cphSetCellParamIgnoreForCheck

```
cphSetCellParamIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be ignored when running *Check Against Source* on the specified logical cell in the specified physical configuration. Mismatches for any of the listed parameters are not reported by the check. The list inherits all the parameter names from the *Ignore for generation* list.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_listOfParamNames	List of parameter names to be ignored. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be ignored was set.
nil	The list of parameter names to be ignored was not set.

Example

```
cphSetCellParamIgnoreForCheck(physConfigID "cph" "nand" "l w")
```

cphSetCellParamIgnoreForGen

```
cphSetCellParamIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be ignored when generating a layout for the specified logical cell in the specified physical configuration. The specified parameters are ignored by the commands; *Generate All From Source*, *Generate Selected From Source*, *Generate Clones*, *Check Against Source*, *Update Components And Nets*, *Update Layout Parameters*, *Update Schematic Parameters*.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_listOfParamNames	List of parameter names to be ignored. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be ignored was set.
nil	The list of parameter names to be ignored was not set.

Example

```
cphSetCellParamIgnoreForGen(physConfigID "cph" "nand" "l w")
```

cphSetCellParamNameMapping

```
cphSetCellParamNameMapping(
     g_physConfigID
     t_logLib
     t_logCell
     t_paramMapping
     )
     => t / nil
```

Description

Defines the mapping between parameter names in the schematic and layout views of the specified cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_paramMapping	String specifying how parameter names are mapped. For example, `l L ; w W ;` Maps schematic parameters `l` and `w` to layout parameters `L` and `W` respectively.

Value Returned

`t`	Parameter name mapping was set.
`nil`	Parameter name mapping was not set.

Example

```
cphSetCellParamNameMapping(physConfigID "cph" "nand" "l L; w W;")
```

cphSetCellParamToCheck

```
cphSetCellParamToCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be checked when running *Check Against Source* on the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_listOfParamNames	List of parameter names to be checked. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be checked was set.
nil	The list of parameter names to be checked was not set.

Example

```
val=cphSetCellParamToCheck(physConfigID "cph" "nand" "l w")
```

cphSetCellRemoveDevice

```
cphSetCellRemoveDevice(
     g_physConfigID
     t_logLib
     t_logCell
     t_removeDevice
     )
     => t / nil
```

Description

Sets the *Remove device* rule for the specified logical cell in the specified physical configuration. A remove device rule causes parasitic devices to be ignored by merging nets connected to the terminals of a single instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

t_removeDevice	String specifying a *Remove device* rule.
	For example, for a resistor with terminal names PLUS and MINUS,
	■ `(short(PLUS MINUS))` would short the terminals.
	■ `(short(PLUS MINUS) funcR(r))` would short the terminals only if the user-defined SKILL function `funcR` returns non-nil.
	A sample `funcR` for the above case would be as follows.
	`procedure( funcR(r)`
	`  if(r<100 then t`
	`  else nil)`
	`)`

Value Returned

`t`	*Remove device* rule was set.
`nil`	*Remove device* rule was not set.

Example

For a cell with terminal names PLUS and MINUS, the following example would short the terminals.

```
cphSetCellRemoveDevice(physConfigID "cph" "nand" "(short(PLUS MINUS))")
```

cphSetCellRounding

```
cphSetCellRounding(
     g_physConfigID
     t_logLib
     t_logCell
     t_roundingRule
     )
     => t / nil
```

Description

Specifies how the value of a specified parameter on a logical cell is rounded when it is evaluated by Layout XL. This property is typically used in conjunction with the `lxDeviceWidth` parameter to prevent folded devices from becoming off-grid.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_roundingRule	String specifying a *Rounding* rule. For example, the following setting rounds the value of `w` to the closest multiple of 0.05 microns. `(w 0.05 round)`

Value Returned

`t`	*Rounding* rule was set.
`nil`	*Rounding* rule was not set.

Example

```
cphSetCellRounding(physConfigID "cph" "nand" "(w 0.05 round)")
```

Related Topics

Rounding Parameter Values

cphSetCellSFactorSplit

```
cphSetCellSFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     g_boolean
     )
     => t / nil
```

Description

Sets the *Split sfactored devices* option to `t` or `nil` for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_boolean	Boolean specifying the value of the *Split sfactored devices* option.

Value Returned

`t`	The *Split sfactored devices* option was set.
`nil`	The *Split sfactored devices* option was not set.

Example

```
cphSetCellSFactorSplit(physConfigID "cph" "nand" t)
```

cphSetCellTermIgnoreForCheck

```
cphSetCellTermIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_listOfTermNames
     )
     => t / nil
```

Description

Defines the list of terminals to be ignored when running *Check Against Source* for the specified logical cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_listOfTermNames	List of terminal names to be ignored. Separate each terminal name with a space and enclose the list in quotation marks.

Value Returned

t	The list of terminals to be ignored was set.
nil	The list of terminals to be ignored was not set.

Example

```
cphSetCellTermIgnoreForCheck(physConfigID "cph" "nand" "S D")
```

cphSetCellTermIgnoreForGen

```
cphSetCellTermIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_listOfTermNames
     )
     => t / nil
```

Description

Defines the list of terminals to be ignored when generating a layout for the specified logical cell in the specified physical configuration. The specified terminals are ignored by the commands: *Generate All From Source*, *Generate Selected From Source*, *Generate Clones*, *Check Against Source*, *Update Components And Nets*, *Update Layout Parameters*, *Update Schematic Parameters*.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_listOfTermNames	List of terminal names to be ignored. Separate each terminal name with a space and enclose the list in quotation marks.

Value Returned

t	The list of terminals to be ignored was set.
nil	The list of terminals to be ignored was not set.

Example

```
cphSetCellTermIgnoreForGen(physConfigID "cph" "nand" "S D")
```

cphSetCellTermNameMapping

```
cphSetCellTermNameMapping(
     g_physConfigID
     t_logLib
     t_logCell
     t_termMapping
     )
     => t / nil
```

Description

Defines the mapping between terminal names in the schematic and layout views of the specified cell in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_termMapping	String specifying how terminal names are mapped. For example, `B G ; X D ;` Maps schematic terminals `B` and `X` to layout terminals `G` and `D` respectively.

Value Returned

`t`	Terminal name mapping was set.
`nil`	Terminal name mapping was not set.

Example

```
cphSetCellTermNameMapping(physConfigID "cph" "nand" "B G ; X D ;")
```

cphSetCellVLGen

```
cphSetCellVLGen(
     g_physConfigID
     t_logLib
     t_logCell
     g_boolean
     )
     => t / nil
```

Description

Sets the VLGen property mapped to a specified logical cell in the physical configuration view associated with the given ID. The cell physical binding is set using the logical library name, logical cell name, and view *layout*.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_boolean	Boolean value specifying the VLGen property. Valid Values: `t` or `nil`.

Value Returned

`t`	The VLGen property has been set on the specified cell.
`nil`	The VLGen property has not been set on the specified cell.

Example

```
cphSetCellVLGen(physConfigID "cph" "nand2" t)
```

Additional Information

If the VLGen CPH property is already defined for the specified cell and then the `cphSetCellVLGen()` SKILL function is set to `nil`, the VLGen property in CPH will also be set to `nil` to indicate that the logical cell is no longer a VLGen. However, the physical binding associated with the cell in CPH is retained. To completely remove the VLGen property

associated with the cell, clear the physical binding in CPH and then run the `cphDeleteCellVLGen` SKILL command.

Related Topics

cphDeleteCellVLGen

cphSetInstFingerSplit

```
cphSetInstFingerSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     g_value
     )
     => t / nil
```

Description

Sets the *Split fingered devices* option for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Split fingered devices* attribute was set.
`nil`	*Split fingered devices* attribute was not set.

Example

```
cphSetInstFingerSplit(physConfigID "cph" "TopCell" "schematic" "I0" t)
```

cphSetInstIgnoreForCheck

```
cphSetInstIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     g_value
     )
     => t / nil
```

Description

Sets the *Ignore for check* option to `t` or `nil` for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Ignore for check* was set.
`nil`	*Ignore for check* was not set.

Example

```
cphSetInstIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0" nil)
```

cphSetInstIgnoreForGen

```
cphSetInstIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     g_value
     )
     => t / nil
```

Description

Sets the *Ignore for layout generation* option to `t` or `nil` for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Ignore for layout generation* was set.
`nil`	*Ignore for layout generation* was not set.

Example

```
cphSetInstIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0" nil)
```

cphSetInstMFactorSplit

```
cphSetInstMFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     g_value
     )
     => t / nil
```

Description

Sets the *Split mfactored devices* option to `t` or `nil` for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Split mfactored devices* was set.
`nil`	*Split mfactored devices* was not set.

Example

```
cphSetInstMFactorSplit(physConfigID "cph" "TopCell" "schematic" "I0" t)
```

cphSetInstParamIgnoreForCheck

```
cphSetInstParamIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be ignored when running *Check Against Source* on the specified instance in the specified physical configuration. Mismatches for any of the listed parameters are not reported by the check. The list inherits all of the parameter names from the *Ignore for generation* list.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the where concerned instance is instantiated.
t_inst	Name of the instance.
t_listOfParamNames	List of parameter names to be ignored. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be ignored was set.
nil	The list of parameter names to be ignored was not set.

Example

```
cphSetInstParamIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0" "l w")
```

cphSetInstParamIgnoreForGen

```
cphSetInstParamIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be ignored when generating a layout for the specified instance in the specified physical configuration. The specified parameters are ignored by the following commands; *Generate All From Source*, *Generate Selected From Source*, *Generate Clones*, *Check Against Source*, *Update Components And Nets*, *Update Layout Parameters*, *Update Schematic Parameters*.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
t_listOfParamNames	List of parameter names to be ignored. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

`t`	The list of parameter names to be ignored was set.
`nil`	The list of parameter names to be ignored was not set.

Example

```
cphSetInstParamIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "I0" "l w")
```

cphSetInstParamToCheck

```
cphSetInstParamToCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be checked when running *Check Against Source* on the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.
t_listOfParamNames	List of parameter names to be checked. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be checked was set.
nil	The list of parameter names to be checked was not set.

Example

```
cphSetInstParamToCheck(physConfigID "cph" "TopCell" "schematic" "I0" "l w")
```

cphSetInstRemoveDevice

```
cphSetInstRemoveDevice(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_removeDevice
     )
     => t / nil
```

Description

Specifies a *Remove device* rule for the specified instance in the specified physical configuration. A remove device rule causes parasitic devices to be ignored by merging nets connected to the terminals of a single instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.

t_removeDevice	String specifying a *Remove device* rule.
	For example, for a resistor with terminal names PLUS and MINUS,
	■ `(short(PLUS MINUS))` would short the terminals.
	■ `(short(PLUS MINUS) funcR(r))` would short the terminals only if the user-defined SKILL function `funcR` returns non-nil.
	A sample `funcR` for the above case would be as follows.
	`procedure( funcR(r)`
	`  if(r<100 then t`
	`  else nil)`
	`)`

Value Returned

`t`	*Remove device* rule was set.
`nil`	*Remove device* rule was not set.

Example

For an instance with terminal names PLUS and MINUS, the following example would short the terminals.

```
cphSetInstRemoveDevice(physConfigID "cph" "TopCell" "schematic" "(short(PLUS
MINUS))")
```

cphSetInstRounding

```
cphSetInstRounding(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_roundingRule
     )
     => t / nil
```

Description

Specifies how the value of a specified parameter on an instance is rounded when it is evaluated by Layout XL. This property is typically used in conjunction with the `lxDeviceWidth` parameter to prevent folded devices from becoming off-grid.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
t_roundingRule	String specifying a *Rounding* rule. For example, the following setting rounds the value of `w` to the closest multiple of 0.05 microns. `(w 0.05 round)`

Value Returned

`t`	*Rounding* rule was set.
`nil`	*Rounding* rule was not set.

Example

```
cphSetInstRounding(physConfigID "cph" "TopCell" "schematic" "(w 0.05 round)")
```

Related Topics

Rounding Parameter Values

cphSetInstSFactorSplit

```
cphSetInstSFactorSplit(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     g_boolean
     )
     => t / nil
```

Description

Sets the *Split sfactored devices* option to `t` or `nil` for the specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical view.
t_logCell	Name of the cell containing the logical view.
t_logView	Name of the logical view.
t_inst	Name of the instance.
g_boolean	Boolean specifying the value of the *Split sfactored devices* option.

Value Returned

`t`	The *Split sfactored devices* option was set.
`nil`	The *Split sfactored devices* option was not set.

Example

```
cphSetInstSFactorSplit(physConfigID "cph" "TopCell" "schematic" "I0" t)
```

cphSetInstTermIgnoreForCheck

```
cphSetInstTermIgnoreForCheck(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_listOfTermNames
     )
     => t / nil
```

Description

Defines the list of terminals to be ignored by when running *Check Against Source* for a specified instance in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
t_listOfTermNames	List of terminal names to be ignored. Separate each terminal name with a space and enclose the list in quotation marks.

Value Returned

t	List of terminals to be ignored was set.
nil	List of terminals to be ignored was not set.

Example

```
cphSetInstTermIgnoreForCheck(physConfigID "cph" "TopCell" "schematic" "I0" "S D")
```

cphSetInstTermIgnoreForGen

```
cphSetInstTermIgnoreForGen(
     g_physConfigID
     t_logLib
     t_logCell
     t_logView
     t_inst
     t_listOfTermNames
     )
     => t / nil
```

Description

Defines the list of terminals to be ignored when generating a layout for the specified instance in the specified physical configuration. The specified terminals are ignored by the commands: *Generate All From Source*, *Generate Selected From Source*, *Generate Clones*, *Check Against Source*, *Update Components And Nets*, *Update Layout Parameters*, *Update Schematic Parameters*.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library where concerned instance is instantiated.
t_logCell	Name of the cell where concerned instance is instantiated.
t_logView	Name of the logical view where concerned instance is instantiated.
t_inst	Name of the instance.
t_listOfTermNames	List of terminal names to be ignored. Separate each terminal name with a space and enclose the list in quotation marks.

Value Returned

`t`	List of terminals to be ignored was set.
`nil`	List of terminals to be ignored was not set.

Example

```
cphSetInstTermIgnoreForGen(physConfigID "cph" "TopCell" "schematic" "S D")
```

cphSetOccurFingerSplit

```
cphSetOccurFingerSplit(
     g_physConfigID
     t_path
     g_value
     )
     => t / nil
```

Description

Sets the *Split fingered devices* option for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Split fingered devices* attribute was set.
`nil`	*Split fingered devices* attribute was not set.

Example

```
cphSetOccurFingerSplit(physConfigID "I0/N0" nil)
```

cphSetOccurIgnoreForCheck

```
cphSetOccurIgnoreForCheck(
     g_physConfigID
     t_path
     g_value
     )
     => t / nil
```

Description

Sets the *Ignore for check* option to `t` or `nil` for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Ignore for check* was set.
`nil`	*Ignore for check* was not set.

Example

```
cphSetOccurIgnoreForGen(physConfigID "I0/N0" t)
```

cphSetOccurIgnoreForGen

```
cphSetOccurIgnoreForGen(
     g_physConfigID
     t_path
     g_value
     )
     => t / nil
```

Description

Sets the *Ignore for layout generation* option to `t` or `nil` for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Ignore for layout generation* was set.
`nil`	*Ignore for layout generation* was not set.

Example

```
cphSetOccurIgnoreForGen(physConfigID "I0/N0" t)
```

cphSetOccurMFactorSplit

```
cphSetOccurMFactorSplit(
     g_physConfigID
     t_path
     g_value
     )
     => t / nil
```

Description

Sets the *Split mfactored devices* option to `t` or `nil` for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_value	Valid Values: `t` or `nil`.

Value Returned

`t`	*Split mfactored devices* was set.
`nil`	*Split mfactored devices* was not set.

Example

```
cphSetOccurMFactorSplit(physConfigID "I0/N0" nil)
```

cphSetOccurParamIgnoreForCheck

```
cphSetOccurParamIgnoreForCheck(
     g_physConfigID
     t_path
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be ignored when running *Check Against Source* on the specified occurrence in the specified physical configuration. Mismatches for any of the listed parameters are not reported by the check. The list inherits all of the parameter names from the *Ignore for generation* list.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_listOfParamNames	List of parameter names to be ignored. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be ignored was set.
nil	The list of parameter names to be ignored was not set.

Example

```
cphSetOccurParamIgnoreForCheck(physConfigID "I0/N0" "l w")
```

cphSetOccurParamIgnoreForGen

```
cphSetOccurParamIgnoreForGen(
     g_physConfigID
     t_path
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be ignored when generating a layout for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_listOfParamNames	List of parameter names to be ignored. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be ignored was set.
nil	The list of parameter names to be ignored was not set.

Example

```
cphSetOccurParamIgnoreForGen(physConfigID "I0/N0" "l w")
```

Additional Information

The specified parameters are ignored by the following commands; *Generate All From Source*, *Generate Selected From Source*, *Generate Clones*, *Check Against Source*, *Update Components And Nets*, *Update Layout Parameters*, *Update Schematic Parameters*.

The following parameters are always ignored.

```
instancesLastChanged          lxPlacementStatus
instNamePrefix                lxRounding
lxIgnoreParamForCAS           lxStopList
lxIgnoredParams               lxTimeStamp
lxMFactorNum                  lxUseCell
lxParamsToIgnore              pin#
lxParamsToIgnoreForCheck      posi
```

cphSetOccurParamToCheck

```
cphSetOccurParamToCheck(
     g_physConfigID
     t_path
     t_listOfParamNames
     )
     => t / nil
```

Description

Defines the list of parameter names to be checked when running *Check Against Source* on the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_listOfParamNames	List of parameter names to be checked. Separate each parameter name with a space and enclose the list in quotation marks.

Value Returned

t	The list of parameter names to be checked was set.
nil	The list of parameter names to be checked was not set.

Example

```
cphSetOccurParamToCheck(physConfigID "I0/N0" "l w")
```

cphSetOccurRemoveDevice

```
cphSetOccurRemoveDevice(
     g_physConfigID
     t_path
     t_removeDevice
     )
     => t / nil
```

Description

Specifies a *Remove device* rule for the specified occurrence in the specified physical configuration. A remove device rule causes parasitic devices to be ignored by merging nets connected to the terminals of a single instance.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.

t_removeDevice	String specifying a *Remove device* rule.
	For example, for a resistor with terminal names PLUS and MINUS,
	■ `(short(PLUS MINUS))` would short the terminals.
	■ `(short(PLUS MINUS) funcR(r))` would short the terminals only if the user-defined SKILL function `funcR` returns non-nil.
	A sample `funcR` for the above case would be as follows.
	`procedure( funcR(r)`
	`  if(r<100 then t`
	`  else nil)`
	`)`

Value Returned

`t`	*Remove device* rule was set.
`nil`	*Remove device* rule was not set.

Example

For a resistor with terminal names PLUS and MINUS, the following example would short the terminals.

```
cphSetOccurRemoveDevice(physConfigID "I0/N0" "(short(PLUS MINUS))")
```

cphSetOccurRounding

```
cphSetOccurRounding(
     g_physConfigID
     t_path
     t_roundingRule
     )
     => t / nil
```

Description

Specifies how the value of a specified parameter on a specific occurrence is rounded when it is evaluated by Layout XL. This property is typically used in conjunction with the `lxDeviceWidth` parameter to prevent folded devices from becoming off-grid.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_roundingRule	String specifying a *Rounding* rule. For example, the following setting rounds the value of `w` to the closest multiple of 0.05 microns. `(w 0.05 round)`

Value Returned

`t`	*Rounding* rule was set.
`nil`	*Rounding* rule was not set.

Example

```
cphSetOccurRounding(physConfigID "I0/N0" "(w 0.05 round)")
```

Related Topics

Rounding Parameter Values

cphSetOccurSFactorSplit

```
cphSetOccurSFactorSplit(
     g_physConfigID
     t_path
     g_boolean
     )
     => t / nil
```

Description

Sets the *Split sfactored devices* option to `t` or `nil` for the specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
g_boolean	Boolean specifying the value of the *Split sfactored devices* option.

Value Returned

`t`	The *Split sfactored devices* option was set.
`nil`	The *Split sfactored devices* option was not set.

Example

```
cphSetOccurSFactorSplit(physConfigID "I0/N0" nil)
```

cphSetOccurTermIgnoreForCheck

```
cphSetOccurTermIgnoreForCheck(
     g_physConfigID
     t_path
     t_listOfTermNames
     )
     => t / nil
```

Description

Defines the list of terminals to be ignored by when running *Check Against Source* for a specified occurrence in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_listOfTermNames	List of terminal names to be ignored. Separate each terminal name with a space and enclose the list in quotation marks.

Value Returned

t	List of terminals to be ignored was set.
nil	List of terminals to be ignored was not set.

Example

```
cphSetOccurTermIgnoreForCheck(physConfigID "I0/N0" "S D")
```

cphSetOccurTermIgnoreForGen

```
cphSetOccurTermIgnoreForGen(
     g_physConfigID
     t_path
     t_listOfTermNames
     )
     => t / nil
```

Description

Defines the list of terminals to be ignored when generating a layout for the specified occurrence in the specified physical configuration. The specified parameters are ignored by the commands: *Generate All From Source*, *Generate Selected From Source*, *Generate Clones*, *Check Against Source*, *Update Components And Nets*, *Update Layout Parameters*, *Update Schematic Parameters*.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_path	Path to the specific occurrence of an instance.
t_listOfTermNames	List of terminal names to be ignored. Separate each terminal name with a space and enclose the list in quotation marks.

Value Returned

t	List of terminals to be ignored was set.
nil	List of terminals to be ignored was not set.

Example

```
cphSetOccurTermIgnoreForGen(physConfigID "I0/N0" "S D")
```

Component Types Attributes Pane Functions

Use the functions described in this section to create and delete component types, and to assign cells to component types.

ctAddCellToCompTypeGroup

ctCreateCompTypeGroup

ctDeleteCellFromCompTypeGroup

ctDeleteCompTypeGroup

ctGetCellCompTypeGroup

ctGetCellCompTypeGroups

ctGetCompTypeCells

ctGetCompTypeGroupAttr

ctGetCompTypeNames

ctSetCompTypeGroupAttr

Related Topics

Component Types Attributes

ctAddCellToCompTypeGroup

```
ctAddCellToCompTypeGroup(
     g_physConfigID
     t_physLib
     t_physCell
     t_ctgName
     )
     => t / nil
```

Description

Assigns a specified physical cell to a component type group in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_physLib	Name of the library containing the physical cell to be assigned.
t_physCell	Name of the physical cell to be assigned.
t_ctgName	Name of the component type group.

Value Returned

t	The physical cell was assigned to the specified component type group.
nil	The physical cell was not assigned.

Example

This example assigns the spnnmos cell in library cph to a component type group called nmos.

```
ctAddCellToCompTypeGroup(physConfigID "cph" "spnnmos" "nmos")
```

ctCreateCompTypeGroup

```
ctCreateCompTypeGroup(
     g_physConfigID
     t_ctgName
     t_deviceType
     t_widthParamName
     t_orientationParamName
     t_drainTermName
     t_gateTermName
     t_sourceTermName
     t_bulkTermName
     t_activeLayers
     f_foldingThresh
     t_views
     )
     => t / nil
```

Description

Creates a component type group with the specified attributes and stores it in the physical configuration associated with the given ID. If the component type group exists already, it is updated with the specified attribute values.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_ctgName	Name of the component type group.
t_deviceType	Identifies the type of devices assigned to the component type group and used by folding, chaining, and assisted MOS and standard cell operations in Layout XL and the custom digital placer. Valid Values: `PMOS`, `NMOS`, `STDCELL`, `STDSUBCONT`, or `FILLER`. **Note:** This argument is called *Component class* on the graphical user interface.
t_widthParamName	Name of the transistor width parameter on the device master cell. You must set this parameter for *NMOS* and *PMOS* component types, even if the devices in the component type will not be folded.
t_orientationParamName	Specifies the device orientation for the component type group.
t_drainTermName	Default name for the drain terminal in the cells assigned to the component type.
t_gateTermName	Default name for the gate terminal in the cells assigned to the component type.
t_sourceTermName	Default name for the source terminal in the cells assigned to the component type.
t_bulkTermName	Default name for the bulk terminal in the cells assigned to the component type.
t_activeLayers	Diffusion layer-purpose pair for NMOS and PMOS devices. The layer you specify must be defined in the technology file and have its material set to one of `pdiff`, `ndiff`, `pwell`, `nwell`, `pplus`, `nplus`, or `diffusion`.
f_foldingThresh	Width beyond which MOS devices are automatically folded. You must set this parameter for *NMOS* and *PMOS* component types even if the devices in the component type will not be folded.

t_views	View name to be used when defining a standard cell substrate contact or filler cell. This parameter is valid only when the component class is set to `STDSUBCONT` or `FILLER`.

Value Returned

`t`	The component type group was created or updated.
`nil`	The component type group was not created.

Example

```
ctCreateCompTypeGroup(physConfigID "nmos" "NMOS" "w" "R90"" "D" "G" "S" "B" "ndiff
drawing" 0.5 "layoutX")
```

ctDeleteCellFromCompTypeGroup

```
ctDeleteCellFromCompTypeGroup(
     g_physConfigID
     t_physLib
     t_physCell
     )
     => t / nil
```

Description

Removes a specified physical cell from its current component type group in the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_physLib	Name of the library containing the physical cell.
t_physCell	Name of the physical cell to be unassigned.

Value Returned

t	The physical cell was unassigned from the specified component type group.
nil	The physical cell was not unassigned.

Example

```
ctDeleteCellFromCompTypeGroup(physConfigID "cph" "spnnmos")
```

ctDeleteCompTypeGroup

```
ctDeleteCompTypeGroup(
     g_physConfigID
     t_ctgName
     )
     => t / nil
```

Description

Deletes a specified component type group from the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_ctgName	Name of the component type group to delete.

Value Returned

t	The component type group was deleted.
nil	The component type group was not deleted.

Example

```
ctDeleteCompTypeGroup(physConfigID "nmos")
```

ctGetCellCompTypeGroup

```
ctGetCellCompTypeGroup(
     g_physConfigID
     t_physLib
     t_physCell
     )
     => t_ctgName / nil
```

Description

Returns the component type group to which a specified physical cell is assigned.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_physLib	Name of the library containing the physical cell.
t_physCell	Name of the physical cell.

Value Returned

t_ctgName	Component type group to which the physical cell is assigned.
nil	The component type group was not returned.

Example

```
ctGetCellCompTypeGroup(physConfigID "cph" "spnnmos")
```

ctGetCellCompTypeGroups

```
ctGetCellCompTypeGroups(
     g_physConfigID
     t_physLib
     t_physCell
     )
     => l_compTypes / nil
```

Description

Returns all the component type groups to which a specified physical cell was assigned.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_physLib	Name of the library containing the physical cell.
t_physCell	Name of the physical cell.

Value Returned

t_ctgName	List of component type groups to which the physical cell was assigned.
nil	The component type groups were not returned.

Example

```
ctgs = ctGetCellCompTypeGroups(cphId "myLib" "nmos")
foreach(ctg ctgs
   params = ctGetCompTypeGroupAttr(cphId ctg "compTypeParams")
)
```

ctGetCompTypeCells

```
ctGetCompTypeCells(
     g_physConfigID
     t_ctgName
     )
     => l_listOfCellNames / nil
```

Description

Returns a list of all the physical cells assigned to the specified component type.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_ctgName	Name of the component type group.

Value Returned

l_listOfCellNames	List of all the physical cells assigned to the component type.
nil	The list was not returned.

Example

```
ctGetCompTypeCells(physConfigID "nmos")
```

ctGetCompTypeGroupAttr

```
ctGetCompTypeGroupAttr(
     g_physConfigID
     t_ctgName
     t_attrName { deviceType | activeLayer | width | foldingThreshold
          | views | drain | gate | source | bulk }
     )
     => g_value / nil
```

Description

Returns the value of a specified attribute in the component type group.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_ctgName	Name of the component type group.
t_attrName	Name of the attribute whose value you want to see. Enclose the name in quotation marks. Valid Values: `deviceType`, `activeLayer`, `width`, `foldingThreshold`, `views`, `drain`, `gate`, `source`, and `bulk`. **Note:** `deviceType` is called *Component class* on the graphical user interface.

Value Returned

g_value	Value of the named attribute. For details of the valid values for each attribute, see `ctCreateCompTypeGroup`.
`nil`	The attribute value was not returned.

Example

```
val=ctGetCompTypeGroupAttr(physConfigID "nmos" "foldingThreshold")
```

Related Topics

ctCreateCompTypeGroup

ctGetCompTypeNames

```
ctGetCompTypeNames(
     g_physConfigID
     )
     => l_listOfNames / nil
```

Description

Returns a list of component type group names for the physical configuration associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

l_listOfNames	List of component type group names for the specified physical configuration view.
nil	The list of names was not returned.

Example

```
names=ctGetCompTypeNames(physConfigID)
```

ctSetCompTypeGroupAttr

```
ctSetCompTypeGroupAttr(
     g_physConfigID
     t_ctgName
     t_attrName { deviceType | activeLayer | width | length | splitParam |
     orientation | edgeType | numFins | widthPerFin | foldingThresholdFins |
     foldingThreshold | foldingMin | foldingStep | views | drain | gate | source |
     bulk }
     g_value
     )
     => t / nil
```

Description

Sets an attribute by name in the component type.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_ctgName	Name of the component type group.
t_attrName	Name of the attribute you want to set. Enclose the name in quotation marks. Valid Values: `deviceType, activeLayer, width, length, splitParam, orientation, edgeType, numFins, widthPerFin, foldingThresholdFins, foldingThreshold, foldingMin, foldingStep, views, drain, gate, source, and bulk.` **Note:** `deviceType` is called *Component class* on the graphical user interface.
g_value	Value for the attribute. For details of the valid values for each attribute, see `ctCreateCompTypeGroup`.

Value Returned

`t`	The attribute value was set.
`nil`	The attribute value was not set.

Example

```
ctSetCompTypeGroupAttr(physConfigID "nmos" "foldingThreshold" 2)
```

Related Topics

ctCreateCompTypeGroup

Soft Block Attributes Pane Functions

Use the functions described in this section to set and retrieve attributes of cells, instances, and occurrences in the soft block table views.

cphSbAddIOPin

cphSbDisplayAllIOPinsInfo

cphSbDefineCovObstruction

cphSbDefineIOPinLabelFlag

cphSbDefineObstruction

cphSbDefineSoftBlock

cphSbDelCovObstruction

cphSbDelObstruction

cphSbDelIOPin

cphSbDelIOPinById

cphSbDisplayBoundaryInfo

cphSbDisplayCovObstructionInfo

cphSbDisplayIOPinInfo

cphSbDisplayObstruction

cphSbDisplaySoftBlockAttributes

cphSbEditIOPin

cphSbEditIOPinById

cphSbEditSoftBlockAttributes

cphSbGetAllIOPins

cphSbGetFilteredIOPins

cphSbGetIOPinId

cphSbGetIOPinName

cphSbGetSoftBlockId

cphSbGetSoftBlocks

cphSbHasCovObstruction

cphSbIsValidIOPin

cphSbRemoveSoftBlock

cphSbSetPolygonalBoundary

cphSbSetRectangularBoundary

cphSbSetRectangularBoundaryUsingUtil

Related Topics

Soft Block Mode in the CPH Window

cphSbAddIOPin

```
cphSbAddIOPin(
     g_cphId
     { g_sbId | l_physLCV }
     t_termName
     t_termType
     [ ?lpp txl_layerPurpose ]
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?number x_number ]
     [ ?criticality x_criticality ]
     [ ?sigType t_sigType ]
     )
     => x_pinID / nil
```

Description

Adds a new pin for the specified soft block and associates the pin with the specified attributes.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId | *l_PhysLCV*	The soft block ID or a list of ("lib" "cell" "view") triplets
t_termName	Terminal name corresponding to the pin
t_termType	Terminal Type of the Pin. Valid terminal types are: "input", "output", "inputoutput", "switch", "jumper", "unused", and "tristate".
?lpp *"txl_layerPurpose"*	Layer Purpose Pair. Use one of the following arguments to identify the layer purpose pair. **Text:** "layerName" – The default purpose is "drawing" “layerName purposeName” **Number:** layerNum – The default purpose is "drawing" **List:** list (“layerName” “purposeName”) list (layerNum, ”purposeName”)
?width *"n_width"*	Width of the pin
?height *"n_height"*	Height of the pin
?number *"x_number"*	Number of `pinFigs` to be created on the pin. The maximum number of `pinFigs` that can be created is 20.
?criticality *"x_criticality"*	Criticality of the terminal (-128 to 128)
?sigType *"t_sigType"*	Signal type of the terminal Valid signal types are: "signal", "ground", "supply", "clock", "analog", "tieOff", "tieHi", "tieLo", "scan", and "reset".

Value Returned

x_pinID	Unique ID of the pin that has been added.
nil	The new pin has not been added.

Examples

```
pinID = cphSbAddIOPin(cphId sbId "AB" "output" ?lpp "metal3 drawing" ?width 10.0
?height 10.0 ?sigType "signal")


pinID = cphSbAddIOPin(cphId sbId "AB_x" "input" ?lpp "metal2" ?width 10.0 ?height
10.0 ?sigType "power")


pinID = cphSbAddIOPin(cphId list("lib" "cell" "view") "AB_y" "inputoutput" ?lpp 12
?width 10.0 ?height 10.0 ?sigType "ground")


pinID = cphSbAddIOPin(cphId sbId "AB_z" "switch" ?lpp list("metal3" "grid") ?width
10.0 ?height 10.0 ?sigType "clock")
```

cphSbDisplayAllIOPinsInfo

```
cphSbDisplayAllIOPinsInfo(
     g_cphId
     { g_sbId | l_physLCV }
     )
     => t / nil
```

Description

Displays pin information for all pins that match the specified `cphID` and soft block ID.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId | *l_physLCV*	The soft block ID or a list of ("lib" "cell" "view") triplets.

Value Returned

`t`	The command was successful. Pin information such as the pin ID, Term Name, TermType, LPP, Width, Height, Number, Criticality, and SigType for all pins corresponding to the specified physical configuration ID and soft Block ID is displayed in the CIW.
`nil`	The command was unsuccessful.

Examples

```
cphSbDisplayAllIOPinsInfo(cphId sbId)
cphSbDisplayAllIOPinsInfo(cphId list("lib" "cell" "view"))
```

cphSbDefineCovObstruction

```
cphSbDefineCovObstruction(
     g_cphId
     g_sbId | l_physLCV
     tx_layer
     g_allowPGNet
     )
     => t / nil
```

Description

Creates a cover obstruction in the specified soft block (*g_sbId*) or physical cellView (*l_physLCV*). The cover obstruction restricts routing on the specified layer and all lower layers.

Arguments

g_cphId	ID of the physical configuration cellview.
g_sbId / physLCV	
	The soft block ID or a list of ("lib" "cell" "view") triplets.
tx_layer	Layer on which the cover obstruction is created.
g_allowPGNet	Specifies whether power/ground nets are allowed on the blockage. Valid values are `t` and `nil`.

Value Returned

`t`	The cover obstruction was created.
`nil`	The command was unsuccessful.

Example

Creates a cover obstruction on the specified soft block, which covers metal 1, metal 2, metal 3, and metal 4:

```
cphSbDefineCovObstruction(cphId sbId ”metal 4” t)
```

cphSbDefineIOPinLabelFlag

```
cphSbDefineIOPinLabelFlag(
     g_cphId
     { g_sbId | l_PhysLCV }
     g_createLabel
     )
     => t / nil
```

Description

Creates I/O pin label.

Arguments

g_cphId	ID of the physical configuration cellview
*g_sbId	l_PhysLCV*	ID of the soft block or a list of ("lib" "cell" "view") triplets.
g_createLabel	When set to `t`, labels are created.

Value Returned

`t`	I/O pin labels were created.
`nil`	The command was unsuccessful.

Example

```
cphSbDefineIOPinLabelFlag(cphId sbid t)
```

cphSbDefineObstruction

```
cphSbDefineObstruction(
     g_cphId
     { g_sbId | l_physLCV }
     t_obstructionType
     [ ?topOffset  n_topOffset ]
     [ ?bottomOffset n_BottomOffset ]
     [ ?leftOffset n_LeftOffset ]
     [ ?rightOffset  n_RightOffset ]
     [ ?layerName t_layerName ]
     )
     => t / nil
```

Description

Creates placement, routing, or other obstructions based on the `obstructionType` argument. All conditions that are enforced for these obstructions through GUI-based flow are applicable here. For example, only one placement blockage is applied per soft block. If a second placement blockage is applied through this API, a warning is issued.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets
t_obstructionType	Type of obstruction to be added. Valid values are: "placement", "routing", "fill", "slot", "pin", "feedthru", and "screen".
?topOffset *n_topOffset*	Top offset of the obstruction
?bottomOffset *n_BottomOffset*	Bottom offset of the obstruction
?leftOffset *n_LeftOffset*	Left offset of the obstruction
?rightOffset *n_RightOffset*	Right offset of the obstruction
?layerName *t_layerName*	Layer name. This argument is ignored when the `obstructionType` is “placement”.

Value Returned

`t`	The obstruction was created.
`nil`	The command was unsuccessful.

Example

```
cphSbDefineObstruction(cphId sbId "routing" ?topOffset 10 ?bottomOffset 10
?rightOffset 10 ?leftOffset 10 ?layerName "metal1")
```

cphSbDefineSoftBlock

```
cphSbDefineSoftBlock(
     g_cphId
     t_PhysLibName
     t_PhysCellName
     t_PhysViewName
     [ ?logLib   t_logicalLibName ]
     [ ?logCell t_logicalCellName ]
     [ ?logView t_logicalViewName ]
     [ ?viewList t_inhViewList ]
     [ ?stopList  t_inhStopViewList ]
     [ ?bType t_btype ]
     [ ?type t_type  ]
     )
     => x_softBlockID / nil
```

Description

Defines a soft block for a given physical and logical `lib:cell:view`. The initial I/O pins for the soft block are automatically generated and attributes are set as per the `.cdsenv` environment variables.

Arguments

g_cphId — ID of the physical configuration cellview.

t_PhysLibName, t_PhysCellName, t_PhysViewName

The physical ("lib" "cell" "view") to which the logical ("lib" "cell" "view") needs to be associated.

?logLib *t_logicalLibName,* ?logCell *t_logicalCellName,* ?logView *t_logicalViewName*

The logical ("lib" "cell" "view") from which soft block attributes need to be retrieved.

?viewList *t_inhViewList*

The design views that will be used to descend into the hierarchical design until stop views are found. The default value is “schematic symbol”.

?stopList *t_inhStopViewList*

The stop view list. While descending the hierarchical design, the command stops if any of the specified views, such as layout or abstract views, are found in this viewlist.

?bType *t_btype*

Type of block to be generated. Valid values are `"CUSTOM"` and `"DIGITAL"`. The default value is `"CUSTOM"`. This argument is case-insensitive. The block type influences the snapping of PR Boundary and pins during level-1 editing as follows:

`CUSTOM` - Pin edges and PR Boundary are snapped to the manufacturing grid.

`DIGITAL` - Pin centers are snapped to the routing grid and the PRBoundary is snapped to the placement grid.

?type *t_type* — Specifies whether a softMacro or blockBlackBox needs to be generated. By default, a softMacro is generated.

Valid values are `"LAYOUT"`, which is the default value, and `"ABSTRACT"`. This argument is case-insensitive.

Value Returned

x_softBlockID	ID of the soft block that was created
nil	The soft block was not created.

Examples

```
sbId = cphSbDefineSoftBlock(cphId "phys_lib_name" "phys_cell_name"
"phys_view_name" ?logLib "logical_lib_name" ?logCell "logical_cell_name" ?logView
"logical_view_name")


sbId = cphSbDefineSoftBlock(cphId "design" "block"  "layout" ?logLib "design"
?logCell "block"  ?logView "schematic" ?bType "CUSTOM" ?Type "LAYOUT")
```

cphSbDelCovObstruction

```
cphSbDelCovObstruction(
     g_cphId
     { g_sbId | l_physLCV }
     )
     => t / nil
```

Description

Deletes the cover obstruction that is available on the specified soft block (`g_sbId`) or physical cellView (`l_physLCV`). The cover obstruction is removed from the specified layer and all lower layers.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId | *l_physLCV*	The soft block ID or a list of ("lib" "cell" "view") triplets

Value Returned

`t`	The cover obstruction was deleted.
`nil`	The command was unsuccessful.

Example

```
cphSbDelCovObstruction(cphId sbId)
```

cphSbDelObstruction

```
cphSbDelObstruction(
     g_cphId
     { g_sbId | l_physLCV }
     t_obstructionType
     [ ?topOffset  n_topOffset ]
     [ ?bottomOffset n_BottomOffset ]
     [ ?leftOffset n_LeftOffset ]
     [ ?rightOffset n_RightOffset ]
     [ ?layerName t_layerName ]
     )
     => t / nil
```

Description

Deletes the specified obstruction.

Arguments

g_cphId	ID of the physical configuration cellview.
g_sbId	ID of the soft block.
l_PhysLCV	List of ("lib" "cell" "view") triplets.
t_obstructionType	Type of obstruction to be deleted. Valid values are: "placement", "routing", "fill", "slot", "pin", "feedthru", and "screen".

`?topOffset` *n_topOffset*

Top offset of the obstruction.

`?bottomOffset` *n_BottomOffset*

Bottom offset of the obstruction.

`?leftOffset` *n_LeftOffset*

Left offset of the obstruction.

`?rightOffset` *n_RightOffset*

Right offset of the obstruction.

`?layerName` *t_layerName*

Layer name. This argument is ignored when the `obstructionType` is “placement”.

Value Returned

`t`	The obstruction was successfully deleted.
`nil`	The command was unsuccessful.

Example

```
cphSbDelObstruction (cphId sbId "slot" "placement" ?leftOffset 20 ?rightOffset 30
?bottomOffset 50 ?topOffset 2 ?layerName "metal1")
```

cphSbDelIOPin

```
cphSbDelIOPin(
     g_cphId
     { g_sbId | l_physLCV }
     t_termName
     )
     => t / nil
```

Description

Deletes an existing I/O pin from the specified soft block.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId | *l_PhysLCV*	The soft block ID or a list of ("lib" "cell" "view") triplets
t_termName	Name of the net

Value Returned

t	The pin has been deleted.
nil	The pin has not been deleted.

Examples

```
cphSbDelIOPin(cphId sbId "A")

cphSbDelIOPin(cphId list("lib" "cell "view") "B")
```

cphSbDelIOPinById

```
cphSbDelIOPinById(
     g_cphId
     x_pinId
     )
     => t / nil
```

Description

Deletes an existing I/O pin with the specified unique pinId.

Arguments

g_cphId	ID of the physical configuration cellview
x_pinId	Unique ID of the pin that needs to be deleted

Value Returned

t	The pin was successfully deleted.
nil	The command was unsuccessful.

Example

```
cphSbDelIOPinById(cphId pinId)
```

cphSbDisplayBoundaryInfo

```
cphSbDisplayBoundaryInfo(
     g_cphId
     { g_sbId | l_physLCV }
     )
     => t / nil
```

Description

Displays boundary attributes corresponding to the specified soft block

Arguments

g_cphId	ID of the physical configuration cellview.
g_sbId | *l_physLCV*	The soft block ID or a list of ("lib" "cell" "view") triplets.

Value Returned

t	Displays the following boundary attributes of the specified soft block: ■ BoundaryType ■ Area ■ PolygonPoints ■ Aspect Ratio ■ Width ■ Height ■ Utilization ■ Area Mode ■ AreaPerGate ■ GateCount ■ AreaEstCallback
nil	The command was unsuccessful.

Examples

```
cphSbDisplayBoundaryInfo(cphId sbId)

cphSbDisplayBoundaryInfo(cphId list("design" "block" "layout"))
```

cphSbDisplayCovObstructionInfo

```
cphSbDisplayCovObstructionInfo(
     g_cphId
     g_sbId | l_physLCV
     )
     => t / nil
```

Description

Displays information about the cover obstruction that is available on the specified soft block (`g_sbId`) or physical cellView (`l_physLCV`).

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId / physLCV	The soft block ID or a list of ("lib" "cell" "view") triplets

Value Returned

`t`	Information about the cover obstruction was displayed.
`nil`	The command was unsuccessful.

Example

```
cphSbDisplayCovObstructionInfo(cphId sbId)
```

cphSbDisplayIOPinInfo

```
cphSbDisplayIOPinInfo(
     g_cphId
     x_pinId
     )
     =>t / nil
```

Description

Displays all relevant information about the pin that is associated with the specified pin ID.

Arguments

g_cphId	ID of the physical configuration cellview
x_pinId	Unique ID of the pin

Value Returned

t	Displays the soft block (lib:cell:view), Term Name, TermType, Lpp, Width, Height, Number, Criticality, and SigType of the pin with the specified pin ID.
nil	The command was unsuccessful.

Example

```
cphSbDisplayIOPinInfo(cphId pinId)
```

cphSbDisplayObstruction

```
cphSbDisplayObstruction(
     g_cphId
     { g_sbId | l_physLCV }
     [ ?obsType t_ObstructionType ]
     )
     => t / nil
```

Description

Displays information about the specified obstruction.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets
[?obsType t_ObstructionType]	Type of obstruction. Valid values are: "placement", "routing", "fill", "slot", "pin", "feedthru", and "screen".

Value Returned

t	Displays information about the obstruction, such as the obstruction type, offsets, and layer name.
nil	The command was unsuccessful.

Example

```
cphSbDisplayObstruction(cphId sbId ?obsType "routing")
```

cphSbDisplaySoftBlockAttributes

```
cphSbDisplaySoftBlockAttributes(
     g_cphId
     l_PhysLCV | g_sbId
     )
     => t / nil
```

Description

Displays the block type, type, inherited view list, and stop view list attributes of the specified soft block.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets

Value Returned

t	Displays the soft block Physical LCV, logical LCV, block type, type, inherited view list, and stop view list of the soft block that corresponds to the specified soft block ID.
nil	The command was unsuccessful.

Examples

```
cphSbDisplaySoftBlockAttributes(cphId sbId)

cphSbDisplaySoftBlockAttributes(cphId list("design" "block" "layout"))
```

cphSbEditIOPin

```
cphSbEditIOPin(
     g_cphId
     { g_sbId | l_physLCV }
     t_termName
     [ ?lpp txl_lpp ]
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?number x_number ]
     [ ?criticality x_criticality ]
     [ ?sigType t_sigType ]
     )
     => t / nil
```

Description

Edits attributes of existing pins.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets
t_termName	Terminal name corresponding to the pin
?lpp *txl_lpp*	Layer Purpose Pair. Valid formats are text, number, and list. Use one of the following arguments to identify the layer purpose pair. **Text:** "layerName" – The default purpose is "drawing" “layerName purposeName” **Number:** layerNum – The default purpose is "drawing" **List:** list(“layerName” “purposeName”) list(layerNum, ”purposeName”)
?width *n_width*	Width of the Pin
?height *n_height*	Height of the Pin
?number *x_number*	Number of `pinFigs` to be created on the pin. The maximum number of `pinFigs` that can be created is 20.
?criticality *x_criticality*	Criticality of the terminal (-128 to 128)
?sigType *t_sigType*	Signal type of the terminal Valid signal types are: "signal", "ground", "supply", "clock", "analog", "tieOff", "tieHi", "tieLo", "scan", and "reset".

Value Returned

`t`	The pin attributes were changed.
`nil`	The command was unsuccessful.

Examples

```
cphSbEditIOPin(cphId sbId "AB_x"  ?lpp "metal2" ?width 10.0 ?height 10.0 ?sigType
"power")
cphSbEditOPin(cphId list("lib" "cell" "view") "AB_y" ?lpp 12 ?width 10.0 ?height
10.0)
cphSbEditIOPin(cphId sbId "AB_z" ?lpp list("metal3" "grid") ?width 10.0 ?sigType
"clock")
cphSbEditIOPin(cphId sbId "AB" ?lpp "metal1 drawing" ?width 5.0)
```

cphSbEditIOPinById

```
cphSbEditIOPinById(
     g_cphId
     x_pinId
     [ ?lpp txl_lpp ]
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?number x_number ]
     [ ?criticality x_criticality ]
     [ ?sigType t_sigType ]
     )
     => t / nil
```

Description

Edit attributes of the pin with the specified ID.

Arguments

g_cphId	ID of the physical configuration cellview
x_pinID	ID of the pin
?lpp *txl_lpp*	Layer Purpose Pair. Valid formats are text, number, and list. Use one of the following arguments to identify the layer purpose pair. **Text:** "layerName" – The default purpose is "drawing" “layerName purposeName” **Number:** layerNum – The default purpose is "drawing" **List:** list(“layerName” “purposeName”) list(layerNum, ”purposeName”)
?width *n_width*	Width of the Pin
?height *n_height*	Height of the Pin
?number *x_number*	Number of `pinFigs` to be created on the pin. The maximum number of `pinFigs` that can be created is 20.
?criticality *x_criticality*	Criticality of the terminal (-128 to 128)
?sigType *t_sigType*	Signal type of the terminal. Valid values are: signal, ground, supply, clock, analog, tieOff, tieHi, tieLo, scan, and reset.

Value Returned

`t`	The pin attributes were changed.
`nil`	The command was unsuccessful.

Examples

```
cphSbEditIOPinById(cphId 20001 ?lpp "metal2" ?width 10.0 ?height 10.0 ?sigType
"power")
cphSbEditIOPinById(cphId 20002 ?lpp 12 ?width 5.0 ?height 1.5)
cphSbEditIOPinById(cphId 20003 ?width 10.5 ?sigType "clock")
```

cphSbEditSoftBlockAttributes

```
cphSbEditSoftBlockAttributes(
     g_cphId
     { g_sbId | l_physLCV }
     [ ?blockType t_blockType ]
     [ ?type t_type ]
     [ ?viewList t_inhViewList ]
     [ ?stopList t_inhStopViewList ]
     )
     => t / nil
```

Description

Edits the attributes of the soft block with the specified ID.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets
?blockType *t_blockType*	Valid values are "CUSTOM" and "DIGITAL". This argument is case-insensitive.
?type *t_type*	Type of soft abstract generation of the block. Valid values are "LAYOUT" and "ABSTRACT". This argument is case-insensitive.
?viewList *t_inhViewList*	The design views that will be used to descend into the hierarchical design to find stop views
?stopList *t_inhStopViewList*	The stop view list. While descending the hierarchical design, the command stops if any of the specified views, such as layout or abstract views, are found in this viewlist.

Value Returned

t	The soft block attributes were changed.
nil	The command was unsuccessful.

Examples

```
cphSbEditSoftBlockAttributes(cphid list("design" "block1" "layout") ?blockType
"DIGITAL")
cphSbEditSoftBlockAttributes(cphId sbId ?blockType "digital" ?type "ABSTRACT")
```

cphSbGetAllIOPins

```
cphSbGetAllIOPins(
     g_cphId
     { g_sbId | l_physLCV }
     )
     => l_pinID / nil
```

Description

Retrieves a list of pin IDs corresponding to the specified soft block.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets

Value Returned

l_pinID	Displays a list of pin IDs corresponding to the specified soft block
`nil`	The command was unsuccessful.

Examples

```
cphSbGetAllIOPins(cphId sbId)
cphSbGetAllIOPins(cphId list("design" "block" "layout"))
```

cphSbGetFilteredIOPins

```
cphSbGetFilteredIOPins(
     g_cphId
     { g_sbId | l_physLCV }
     [ ?rexTermName  t_regExpTermName ]
     [ ?termType t_termType ]
     [ ?lpp    txl_layerPurpose ]
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?number  x_number ]
     [ ?criticality x_criticality ]
     [ ?sigType t_sigType ]
     )
     => l_pinIds / nil
```

Description

Filters pins as per the given arguments and through regular expressions, and returns a list of pin IDs of all the filtered pins.

Arguments

g_cphId	ID of the physical configuration cellview.
g_sbId	ID of the soft block.
l_PhysLCV	List of ("lib" "cell" "view") triplets.
?rexTermName *t_regExpTermName*	Regular expression.
?termType *t_termType*	Terminal type of the pin. Valid terminal types are: "input", "output", "inputoutput", "switch", "jumper", "unused", and "tristate".
?lpp *txl_layerPurpose*	Layer Purpose Pair. Valid formats are text, number, and list. Use one of the following arguments to identify the layer purpose pair. **Text:** "layerName" – The default purpose is "drawing" "layerName purposeName" **Number:** layerNum – The default purpose is "drawing" **List:** list("layerName" "purposeName") list(layerNum, "purposeName")
?width *"n_width"*	Width of the pin.
?height *"n_height"*	Height of the pin.
?number *"x_number"*	Number of `pinFigs` to be created on the pin. The maximum number of `pinFigs` that can be created is 20.
?criticality *"x_criticality"*	Criticality of the terminal (-128 to 128).

`?sigType` *"t_sigType"*	Signal type of the terminal Valid signal types are: "signal", "ground", "supply", "clock", "analog", "tieOff", "tieHi", "tieLo", "scan", and "reset".

Value Returned

l_pinIds	Displays a list of pin IDs of all the filtered pins.
`nil`	The command was unsuccessful.

Examples

```
pinIds = cphSbGetFilteredIOPins(cphId sbId ?rexTermName "Pin*" ?lpp list("metal6" "pin"))

pinIds = cphSbGetFilteredIOPins(cphId list("design" "block2" "layout") ?lpp list(2 "drawing") ?width 5.0)
```

cphSbGetIOPinId

```
cphSbGetIOPinId(
     g_cphId
     { g_sbId | l_physLCV }
     t_termName
     )
     => x_pinId / nil
```

Description

Returns the pin ID of the soft block corresponding to the specified terminal name.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List of ("lib" "cell" "view") triplets
t_termName	Name of the terminal

Value Returned

x_pinId	ID of the soft block corresponding to the specified terminal name
nil	The command was unsuccessful.

Examples

```
pinId = cphSbGetIOPinId(cphId sbId "Y")
pinId = cphSbGetIOPinId(cphId list("design" "block" "layout") "Y")
```

cphSbGetIOPinName

```
cphSbGetIOPinName(
     g_cphId
     x_pinID
     )
     => l_termNameAndsoftBlockName / nil
```

Description

Retrieves the I/O pin name corresponding to the specified pin ID.

Arguments

g_cphId	ID of the physical configuration cellview
x_pinID	ID of the pin

Value Returned

l_termNameAndsoft-BlockName	Displays the terminal name and the soft block name of the pin corresponding to the specified pin ID.
nil	The pin does not exist.

Example

```
cphSbGetIOPinName(cphId pinID)
```

cphSbGetSoftBlockId

```
cphSbGetSoftBlockId(
     g_cphId
     t_PhysLibName
     t_PhysCellName
     t_PhysViewName
     )
     => x_softBLockId / nil
```

Description

Returns the ID of the soft block corresponding to the specified physical configuration ID and physical `lib:cell:view`

Arguments

g_cphId	ID of the physical configuration cellview.
t_PhysLibName, t_PhysCellName, t_PhysViewName	The physical ("lib" "cell" "view") of the soft block.

Value Returned

x_softBLockId	ID of the soft block corresponding to the specified `cphId` and physical ("lib" "cell" "view")
`nil`	Soft block does not exist.

Example

```
cphSbGetSoftBlockId(cphId "phys_lib_name" "phys_cell_name" "phys_view_name")
```

cphSbGetSoftBlocks

```
cphSbGetSoftBlocks(
     g_cphId
     )
     => l_softBlockID / nil
```

Description

Returns a list of defined soft blocks corresponding to the `g_cphId`.

Arguments

g_cphId	ID of the physical configuration cellview

Value Returned

l_softBlockID	A list of soft blocks is returned in the following format: `(sbId "physLib:physCell:physView" "logical-Lib:logicalCell:logicalView")`
`nil`	The command was unsuccessful.

Example

```
cphSbGetSoftBlocks(cphId)
```

cphSbHasCovObstruction

```
cphSbHasCovObstruction(
     g_cphId
     g_sbId | l_physLCV
     )
     => t / nil
```

Description

Checks for the presence of a cover obstruction on the specified soft block (*g_sbId*) or physical cellView (*l_physLCV*).

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId / physLCV	The soft block ID or a list of ("lib" "cell" "view") triplets

Value Returned

t	A cover obstruction is available.
nil	The command was unsuccessful.

Example

```
cphSbHasCovObstruction(cphId sbId)
```

cphSbIsValidIOPin

```
cphSbIsValidIOPin(
     g_cphId
     x_pinId
     )
     => t / nil
```

Description

Checks if the pin with the specified pin ID exists or not.

Arguments

g_cphId	ID of the physical configuration cellview
x_pinId	ID of the pin

Value Returned

t	The pin exists.
nil	The pin does not exist.

Example

```
cphSbIsValidIOPin(cphId pinId)
```

cphSbLoadSoftBlocks

```
cphSbLoadSoftBlocks(
     g_cphId
     t_fileName
     )
     => t / nil
```

Description

Loads soft block data from the specified floorplan property file. In case of duplicate soft blocks data entries in the file, the function only considers the first data entry, and ignores subsequent duplicate entries.

Arguments

g_cphId	ID of the physical configuration cellview
t_fileName	Name of the floorplan property file from which soft block data needs to be loaded

Value Returned

`t`	Soft block data was loaded from the specified floorplan property file.
`nil`	Read operation failed. Soft block data was not loaded from the specified floorplan property file.

Example

```
cphSbLoadSoftBlocks(cphId "softBlocks.txt")
```

cphSbRemoveSoftBlock

```
cphSbRemoveSoftBlock(
     g_cphId
     g_sbId
     )
     => t / nil
```

Description

Removes the soft block that is associated with the specified ID.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block that needs to be deleted

Value Returned

t	The soft block was successfully removed.
nil	The soft block was not removed.

Example

```
cphSbRemoveSoftBlock(cphId sbId)
```

cphSbSaveSoftBlocks

```
cphSbSaveSoftBlocks(
     g_cphId
     t_fileName
     [ ?mode t_fileMode ]
     )
     => t / nil
```

Description

Saves soft block data to the specified floorplan property file.

Arguments

g_cphId	ID of the physical configuration cellview
t_fileName	Name of the floorplan property file in which soft block data needs to be stored
?mode *t_fileMode*	Mode in which the floorplan property file needs to be opened. Supported modes are: ■ **"w" - Write mode:** Writes the soft blocks data into the file. Any existing file with the same name will get overwritten by the data. This is the default option. ■ **"a" - Append mode:** Appends the soft blocks data to the existing file. Ensure that soft blocks have been generated from the same logical views, although they can have different physical configuration cellviews.  ***This mode does not check for duplication of soft blocks data while appending the floorplan property file.***

Value Returned

t	Soft block data was written to the specified floorplan property file.
nil	Write operation failed. Soft block data was not written to the specified floorplan property file.

Example

```
cphSbSaveSoftBlocks(cphId "softBlocks.txt")
cphSbSaveSoftBlocks(cphId "softBlocks.txt" ? mode "a")
```

cphSbSetPolygonalBoundary

```
cphSbSetPolygonalBoundary(
     g_cphId
     { g_sbId | l_physLCV }
     l_listOfPolygonPoints
     )
     => t / nil
```

Description

Sets the boundary attributes of soft blocks that are polygonal in shape. You need to specify the boundary vertices as a list of points or x, y coordinates.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId | *l_physLCV*	The soft block ID or the physical ("lib" "cell" "view")
l_listOfPolygon-Points	The vertices of the rectilinear polygon in the form of a list of x, y coordinates

Value Returned

t	The vertices of the polygonal soft block were set.
nil	The vertices of the polygonal soft block were not set.

Examples

```
cphSbSetPolygonalBoundary (cphId sbId list('(0 0) '(0 100) '(100 100) '(100 50)
'(50 50) '(50 0)))
cphSbSetPolygonalBoundary(cphId list("design" "block" "layout") list('(0 0) '(0
9.5) '(80.9 9.5) '(80.9 90.5) '(100.5 90.5) '(100.5 0)))
```

cphSbSetRectangularBoundary

```
cphSbSetRectangularBoundary(
     g_cphId
     { g_sbId | l_physLCV }
     n_width
     n_height
     )
     => t / nil
```

Description

Sets the soft block rectangular boundary with the specified dimensions.

Arguments

g_cphId	ID of the physical configuration cellview
g_sbId	ID of the soft block
l_PhysLCV	List specifying the ("lib" "cell" "view") corresponding to the soft block
n_width	Width of the rectangular boundary
n_height	Height of the rectangular boundary

Value Returned

t	The boundary vertices were set.
nil	The command was unsuccessful.

Examples

```
cphSbSetRectangularBoundary(cphId sbId 700 700)
cphSbSetRectangularBoundary(cphId list("design" "block" "layout") 56.1 78.4)
```

cphSbSetRectangularBoundaryUsingUtil

```
cphSbSetRectangularBoundaryUsingUtil(
     g_cphId
     l_PhysLCV | g_sbId
     n_util
     s_mode
     [ ?width n_width ]
     [ ?height n_height ]
     [ ?aspRatio n_aspRatio ]
     [ ?area n_area ]
     [ ?areaPerGate n_areaPerGate ]
     [ ?gateCount x_gateCount ]
     [ ?areaEstCallback t_areaEstCallback ]
     )
     => t / nil
```

Description

Sets a rectangular boundary for the specified soft block with the specified area utilization. It is mandatory to specify any one of the three optional parameters: width, height, and aspRatio.

Arguments

g_cphId	ID of the physical configuration cellview.
g_sbId	ID of the soft block.
l_PhysLCV	List of ("lib" "cell" "view") triplets.
n_util	Area utilization for the soft block.
s_mode	Mode can take one of following three values to determine the parameters to be used when calculating the area. ■ manual: Only area will be taken along with one of the three optional parameters: width, height, and aspRatio ■ avgAreaPerGate: Only areaPerGate and gateCount will be taken along with one of the three optional parameters: width, height, and aspRatio ■ useEstimator: Only areaEstCallback will be taken along with one of the three optional parameters: width, height, and aspRatio If any of the required parameters is found nil, an error will be thrown.
?width *n_width*	Width of the rectangular soft block.
?height *n_height*	Height of the rectangular soft block.
?aspRatio *n_aspRatio*	Aspect Ratio of the rectangular soft block.
?area *n_area*	Area of the rectangular soft block.
?areaPerGate *n_areaPerGate*	Area per gate of the rectangular soft block.
?gateCount *x_gateCount*	Total gate count of the rectangular soft block.
?areaEstCallback *t_areaEstCallback*	Area estimator callback function.

Value Returned

`t`	The rectangular boundary of the specified soft block has been set with specified area utilization.
`nil`	The command was unsuccessful.

Examples

Using the manual mode:

```
cphSbSetRectangularBoundaryUsingUtil(cphId sbB1 50 'manual ?width 20 ?area 5000)
```

Using the avgAreaPerGate mode:

```
cphSbSetRectangularBoundaryUsingUtil(cphId sbB2 50 'avgAreaPerGate ?height 20
?areaPerGate 100 ?gateCount 50)
```

Using the useEstimator mode:

```
cphSbSetRectangularBoundaryUsingUtil(cphId sbB2 50 'useEstimator ?width 20
?areaEstCallback "estAreaUsingLayoutPR")
```

Visitor Functions

Use the functions described in this section to explore the content of the elaborated hierarchy associated with a specified physical configuration cellview. You control the visit using the cphVisitStart, cphVisitNextNode, and cphVisitStop functions. The remaining functions let you access different attributes of the current node during the visit.

cphVisitedInstance

cphVisitedPath

cphVisitedSwitchMaster

cphVisitedTarget

cphVisitNextNode

cphVisitStart

cphVisitStop

cphVisitedInstance

```
cphVisitedInstance(
     g_physConfigID
     )
     => d_instID / nil
```

Description

Returns the database ID of the instance of the occurrence currently being visited.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

d_instID	Database ID of the instance.
nil	The database ID of the instance was not returned.

Example

```
instId=cphVisitedInstance(physConfigID)
```

cphVisitedPath

```
cphVisitedPath(
     g_physConfigID
     )
     => t_path / nil
```

Description

Returns the path to the occurrence currently being visited.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

t_path	Path to the occurrence currently being visited.
nil	The path was not returned.

Example

```
path=cphVisitedPath(physConfigID)
=> "I0/N0"
```

cphVisitedSwitchMaster

```
cphVisitedSwitchMaster(
     g_physConfigID
     )
     => d_logCellviewID / nil
```

Description

Returns the database ID of the switch master cellview for the occurrence currently visited.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

d_logCellviewID	Database ID of the switch master cellview.
nil	The switch master cellview ID was not returned.

Example

```
cellviewId=cphVisitedSwitchMaster(physConfigID)
```

cphVisitedTarget

```
cphVisitedTarget(
     g_physConfigID
     )
     => d_logCellviewID / nil
```

Description

Returns the database ID of the logical cellview for the occurrence currently being visited.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

d_logCellviewID	Database ID of the logical cellview.
nil	The logical cellview ID was not returned.

Example

```
cellviewId=cphVisitedTarget(physConfigID)
```

cphVisitNextNode

```
cphVisitNextNode(
     g_physConfigID
     )
     => t / nil
```

Description

Moves the visitor to the next occurrence as defined by the start mode specified in `cphVisitStart`. The start mode defines which nodes are visited, but not order in which they are visited.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

`t`	The visitor moved to the next occurrence.
`nil`	The visit is finished.

Example

```
cphVisitNextNode(physConfigID)
```

cphVisitStart

```
cphVisitStart(
     g_physConfigID
     t_startMode { PHYSICAL_LEAF | LOGICAL_LEAF | HIERARCHY_PHYSICAL_LEAF
          | HIERARCHY_LOGICAL_LEAF }
     [ t_path ]
     )
     => t / nil
```

Description

Starts a visit at the specified occurrence in the specified mode. The start mode defines which nodes are visited, but does not specify the order in which they are visited.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_startMode	Specifies which types of nodes are to be visited. ■ PHYSICAL_LEAF: Visits only leaf nodes with physical decoration ■ LOGICAL_LEAF: Visits only leaf nodes ■ HIERARCHY_PHYSICAL_LEAF: Visits all nodes except leaf nodes with no physical decoration ■ HIERARCHY_LOGICAL_LEAF: Visits all nodes **Note:** The start mode does not specify the order in which nodes are visited.
t_path	Path to a specific occurrence of an instance. If you do not specify the path, the visit starts at the root node in the hierarchy.

Value Returned

t	The visitor started.
nil	The visitor did not start.

Example

```
cphVisitStart(physConfigID "PHYSICAL_LEAF")
cphVisitStart(physConfigID "PHYSICAL_LEAF" "I1/I2")
```

cphVisitStop

```
cphVisitStop(
     g_physConfigID
     )
     => t |/ nil
```

Description

Stops the visitor currently running in the specified physical configuration.

Arguments

g_physConfigID	ID of the physical configuration cellview.

Value Returned

`t`	The visitor stopped.
`nil`	The visitor did not stop.

Example

```
cphVisitStop(physConfigID)
```

Virtuoso Parameterized Layout Generator Functions

Use the functions described in this section to set and retrieve information associated with Virtuoso Parameterized Layout Generators.

cphDeleteCellVPLGen

cphGetCellVPLGen

cphGetCellVPLGenParams

cphSetCellVPLGen

cphSetCellVPLGenParams

dbIsVPLGen

dbRegVPLGenCreateCellName

dbUnregVPLGenCreateCellName

cphDeleteCellVPLGen

```
cphDeleteCellVPLGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Deletes the VPLGen mapped to a specified logical cell in the physical configuration view associated with the given ID. The VPLGen property, parameters, and physical binding are deleted and the VPLGen is removed from disk.

Cadence recommends that you do not use this function unless you are sure that you no longer want to use the VPLGen in question. Using this function to delete a VPLGen means that any designs that use that VPLGen will have missing layout instances.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The VPLGen was deleted.
nil	The VPLGen was not deleted.

Example

```
cphDeleteCellVPLGen(physConfigID "cph" "nand2")
```

cphGetCellVPLGen

```
cphGetCellVPLGen(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t / nil
```

Description

Returns the VPLGen property mapped to a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t	The VPLGen property is set on the specified cell.
nil	The VPLGen property is not set on the specified cell.

Example

```
cphGetCellVPLGen(physConfigID "cph" "nand2")
```

cphGetCellVPLGenParams

```
cphGetCellVPLGenParams(
     g_physConfigID
     t_logLib
     t_logCell
     )
     => t_parameters / nil
```

Description

Returns the VPLGen parameters mapped to a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.

Value Returned

t_parameters	Name and values of the VPLGen parameters set on the specified cell.
nil	There are no VPLGen parameters set on the specified cell.

Example

```
cphGetCellVPLGenParams(cph "cph" "nand2")
```

cphSetCellVPLGen

```
cphSetCellVPLGen(
     g_physConfigID
     t_logLib
     t_logCell
     g_boolean
     )
     => t / nil
```

Description

Sets the VPLGen property mapped to a specified logical cell in the physical configuration view associated with the given ID. The cell physical binding is set using the logical library name, logical cell name, and view name `layout`.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
g_boolean	Boolean specifying the VPLGen property to set.

Value Returned

`t`	The VPLGen property was set on the specified cell.
`nil`	The VPLGen property was not set on the specified cell.

Example

```
cphSetCellVPLGen(physConfigID "cph" "nand2" t)
```

cphSetCellVPLGenParams

```
cphSetCellVPLGenParams(
     g_physConfigID
     t_logLib
     t_logCell
     t_parameters
     )
     => t / nil
```

Description

Sets the VPLGen parameters mapped to a specified logical cell in the physical configuration view associated with the given ID.

Arguments

g_physConfigID	ID of the physical configuration cellview.
t_logLib	Name of the library containing the logical cell.
t_logCell	Name of the logical cell.
t_parameters	List of parameter names (and default values for non-CDF parameters), each separated by a semicolon: For example, `"pl 2u;pw 2u;nl 2u;nw 2u"` If the parameter in question is a CDF parameter (like `pw` or `nw` in the example below), you do not need to specify a default value; the default set in the schematic is used. For example, `"pw;nw;a 1"`

Value Returned

`t`	The VPLGen parameters were set on the specified cell.
`nil`	The VPLGen parameters were not set on the specified cell.

Example

```
cphSetCellVPLGenParams(physconfigID "cph" "nand2" "pl 2u;pw 2u;nl 2u;nw 2u")
```

dbIsVPLGen

```
dbIsVPLGen(
     d_cellviewID
     )
     => t / nil
```

Description

Returns true if the design is a VPLGen.

Arguments

d_cellviewID	Database ID of the supermaster cellview.

Value Returned

t	The design is a VPLGen.
nil	The design is not a VPLGen.

Example

```
dbIsVPLGen(cvId)
```

dbRegVPLGenCreateCellName

```
dbRegVPLGenCreateCellName(
     s_functionName
     )
     => t / nil
```

Description

Registers a SKILL function to create unique cell names for VPLGen core layouts.

Arguments

s_functionName	Name of the SKILL function to be registered. The SKILL function must return a string equal to the cell name. It takes two arguments: *cv*: The database ID of the VPLGen supermaster. *params*: A list of lists of string pairs, where the first string in each pair is the parameter name and the second string in each pair is the parameter value. The pairs are sorted alphabetically by parameter name.

Value Returned

`t`	The function was registered.
`nil`	The function was not registered.

Example

```
dbRegVPLGenCreateCellName("VPLGenCellName")
```

Where the SKILL function `VPLGenCellName` is defined as:

```
procedure(VPLGenCellName(cv params)
     let((name)
         name = cv~>cellName
        foreach(param params
             name = strcat(name sprintf(nil "%s%s" car(param) cadr(param)))
         )
         name = pcreReplace(pcreCompile("\\.") name "_" 0)
         name
     )
)
```

dbUnregVPLGenCreateCellName

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

Description

Unregisters the SKILL function used to generate unique cell names for VPLGen core layouts. Once unregistered, Layout XL uses the default naming convention when creating new VPLGen Pcell instances in the layout view.

Arguments

None

Value Returned

`t`	The SKILL function was unregistered.
`nil`	The SKILL function was not unregistered.

Example

```
dbUnregVPLGenCreateCellName()
```

11

Object Display Control Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso object display control.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- odcRegBlockage
- odcRegBoundary
- odcRegCPA
- odcRegInstance
- odcRegisterCustomFunc
- odcRegLabel
- odcRegMarker
- odcRegModgen
- odcRegPin
- odcRegRow
- odcRegRowRegion
- odcRegRuler
- odcRegShapeCircle
- odcRegShapeDonut
- odcRegShapeEllipse
- odcRegShapeMPP
- odcRegShapePath

- odcRegShapePathSeg
- odcRegShapeRect
- odcRegTextDisplay
- odcRegVia
- odcRegVirtualFigGroup

Customizing the Info Balloon

You can customize the information displayed in an Info Balloon box through a text string that can be appended to the default object attribute information. The custom SKILL *odcReg** functions allow you to display custom information about an object when a mouse cursor points at the object.

You can do this by defining your own SKILL extensions. When you pass the name of your SKILL extension as an argument to the appropriate *odcReg** SKILL function, the Info Balloon box on the corresponding object not only displays the object attributes, but also the string specified by you through the SKILL extension. For further details about using these custom SKILL functions, see Using the SKILL Extension.

Defining the SKILL Extension

The value returned by the SKILL extension is a string. The following syntax is used for defining a SKILL extension:

```
procedure(customSKILLEXTName(dbId @optional hierPath)
     ...
     )
```

where:

customSKILLEXTName	The name of the SKILL extension you define. You pass this name as an argument to an appropriate *odcReg** SKILL function.
dbId	Database ID

hierPath

This is an optional argument. It specifies the following hierarchical information as a list:

`(dbInstId memInst row column)`

where, `dbInstId`: The instance ID in the hierarchical path.

`memInst`: The iteration number for instances. This is `0` as it is not applicable in layout.

`row`: The row position in a mosaic instance. This is 0 as Info Balloon is not applicable for a mosaic instance.

`column`: The column position in a mosaic instance. This is 0 as Info Balloon is not applicable for a mosaic instance.

The following is a sample of a user-defined SKILL extension. The input for the SKILL extension should be a single `dbId` and an optional string for hierarchical path. The output to the CIW is the `dbId`.

```
procedure(rectCustomFunc(dbId @optional hierPath)
let((hierStr)
   if(hierPath then
      hierStr = ")"
      foreach(instInfo hierPath
              inst = car(instInfo)
              hierStr = strcat(hierStr "/" inst~>name)
      )
      hierStr = strcat(hierStr ")")
   else
      hierStr = "()"
; need to return a SKILL string
hierStr
); let
)
```

Using the SKILL Extension

To get your SKILL extension working for a specific object, perform the following steps:

1. In the *Setup* tab of the *Dynamic Display* form, enable the *Custom SKILL Function* option for the object.
2. In the CIW, load the required SKILL extension. For information about how to define the SKILL extension, see Defining the SKILL Extension.

3. Register your SKILL extension as a custom SKILL function for the particular object by passing the name of the SKILL extension as an argument to the required *odcReg** SKILL function.

For example, for the SKILL extension `rectCustomFunc` defined in the section Defining the SKILL Extension can be registered in the following way:

```
odcRegShapeRect("Layout-XL" "rectCustomFunc")
```

This will display the hierarchy path of a rectangle in the Info Balloon box when you point at the rectangle in the design. Consider a scenario in which there is a rectangle of a bottom cell in the hierarchy top/middle (`I1`)/bottom (`I2`). The `rectCustomFunc` procedure will append `"I1/I2"` at the end of the Info Balloon text. This avoids the need to descent edit to a certain level of hierarchy and then using `geGetInstHierPath`.

odcRegBlockage

```
odcRegBlockage(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for blockages. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application name. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegBlockage("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegBoundary

```
odcRegBoundary(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for boundaries. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegBoundary("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegCPA

```
odcRegCPA(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for a custom placement area. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegCPA("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegInstance

```
odcRegInstance(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for instances. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegInstance("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegisterCustomFunc

```
odcRegisterCustomFunc(
     t_odcObjType
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for the specified object type.

Arguments

t_odcObjType	The object type for which the `customFunc` will be invoked. Valid values are: (“Rectangle”, ”Polygon”, “Path”, ”MPP”, ”PathSeg”, “Circle”, “Ellipse”, “Donut”, “Label”, “TextDisplay”, “Instance”, “Blockage”, “Row”, “Boundary”, “Via”, “Custom Placement Area”, “Marker”, “Pin”, “Modgen”, “Ruler”)
t_functionText	The name of the function to be executed for objects of specified `odcObjType`.

Value Returned

`t`	The function is successfully registered.
`nil`	The function is not successfully registered.

Examples

To define the SKILL extension to return the custom information string:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegisterCustomFunc("Pin" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegLabel

```
odcRegLabel(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for labels. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegLabel("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegMarker

```
odcRegMarker(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for markers of type *Tool*, *Short Message*, and *Severity* information. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegMarker("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegModgen

```
odcRegModgen(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for modgen. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The name of the function to be executed for modgen.

Value Returned

`t`	The function is successfully registered.
`nil`	The function is not successfully registered.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegModgen("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegPin

```
odcRegPin(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for names and net names of pins. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegPin("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegRow

```
odcRegRow(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for rows. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegRow("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegRowRegion

```
odcRegRowRegion(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for row region in advanced node. The application must be specified. Currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The name of the function to be executed for row region.

Value Returned

`t`	The function is successfully registered.
`nil`	The function is not successfully registered.

Examples

To define the SKILL extension to return the custom information string:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegRowRegion("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegRuler

```
odcRegRow(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for rulers. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The name of the function to be executed for ruler objects.

Value Returned

`t`	The function is successfully registered.
`nil`	The function is not successfully registered.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegRuler("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapeCircle

```
odcRegShapeCircle(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for circles. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

Define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

Register the SKILL extension as a custom SKILL function:

```
odcRegShapeCircle("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapeDonut

```
odcRegShapeDonut(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for donuts. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegShapeDonut("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapeEllipse

```
odcRegShapeEllipse(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for donuts. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegShapeEllipse("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapeMPP

```
odcRegShapeMPP(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for MPPs. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegShapeMPP("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapePath

```
odcRegShapePath(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for paths. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegShapePath("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapePathSeg

```
odcRegShapePathSeg(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for pathSegs. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

Register the SKILL extension as a custom SKILL function:

```
odcRegShapePathSeg("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapePolygon

```
odcRegShapePolygon(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for polygons. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

Register the SKILL extension as a custom SKILL function:

```
odcRegShapePolygon("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegShapeRect

```
odcRegShapeRect(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for rectangles. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegShapeRect("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegTextDisplay

```
odcRegTextDisplay(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for text displays. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Example

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegLabel("Layout-XL" "customFunc")
```

Related Topics

Using the SKILL Extension

odcRegVia

```
odcRegVia(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for vias. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegVia("Layout-XL" "customFunc")
```

odcRegVirtualFigGroup

```
odcRegVirtualFigGroup(
     t_deAppName
     t_functionText
     )
     => t / nil
```

Description

Allows you to define a custom SKILL function to specify your own information to be displayed in the Info Balloon box for virtual figGroups. The application must be specified and currently only `Layout-XL` is supported.

Arguments

t_deAppName	The application. Currently only `Layout-XL` is supported.
t_functionText	The text to be displayed in the Info Balloon box.

Value Returned

`t`	The text is successfully displayed.
`nil`	The text is not successfully displayed.

Examples

To define the SKILL extension:

```
procedure(customFunc(dbId @optional hierPath)
let((skillStr)
printf("=== input parameter: %L\n" dbId)
; need to return a SKILL string
skillStr = "test custom \n SKILL function\n"
skillStr
); let
)
```

To register the SKILL extension as a custom SKILL function:

```
odcRegVirtualFigGroup("Layout-XL" "customFunc")
```

12

Interactive and Assisted Routing Functions

This topic provides a list of Cadence® SKILL functions associated with interactive and associated routing.

Only the functions listed in this topic are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- leFinishTrunk
- leFinishWire
- leHiAddWireVia
- leHiCancelStitch
- leHiCreateBus
- leHiCreateGeometricWire
- leHiCreateStrandedWire
- leHiCreateWire
- leHiP2P
- leHiStitchToLayer
- leWECycleControlWire
- leWECycleSnap
- leWENoSnap
- weAddCustomTransitionMenuItem
- weAutoTwigCycleTrunkViaAlignment
- weCWHoldWidth

- weCycleCutColorVia
- weCyclePatternGravityLayerTransitionType
- weGatherBusWires
- weGetCustomTransitionMenuItems
- weGetPathSegWidth
- weHiCopyRoute
- weHiCycleViaDefDown
- weHiCycleViaDefUp
- weHiEditBus
- weHiInteractiveRouting
- weHiWireTo45
- weRemoveCustomTransitionMenuItem
- weScaleMagnifierOrDecreaseWidth
- weScaleMagnifierOrIncreaseWidth
- weScrollOrCycleDownWireViaAlignment
- weScrollOrCycleUpWireViaAlignment
- weSetPathSegWidth

leFinishTrunk

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

Description

When the *Create Wire* command is active, this function converts the current wire(s) being edited into trunks. It then uses Pin To Trunk routing to completes the connection between the converted trunk(s) and all possible pin targets that are orthogonal to the trunk(s). The default bindkey is `2`.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was not successful or an error occurred.

Examples

The following example will finish the current wire from the last click point and perform Pin To Trunk routing using the current wire as a trunk.

```
leFinishTrunk()
```

Related Topics

Interactive and Assisted Routing Functions

Finishing a Trunk

leFinishWire

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

Description

When the *Create Wire* command is active, this function uses the point-to-point router to complete the current wire or wires being edited from their current end to the nearest viable target.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was not successful or an error occurred.

Examples

The following example completes the current wire from the last click point.

```
leFinishWire()
```

Related Topics

Interactive and Assisted Routing Functions

leHiAddWireVia

```
leHiAddWireVia(
     [ w_windowId ]
     [ t_defaultTarget ]
     [ g_rotate ]
     )
     => t / nil
```

Description

When the *Create – Wire* command is active, this function allows you to place a via on a path or pathSeg or opens the *Select Via* form and select a via. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	The current window ID.
t_defaultTarget	Valid Values: `unique`, `select`, `up`, `down`, `cancel`, `layerName`, `viaName`, `t`, `nil`. ■ `unique` or `t`: Selects the only available via otherwise opens the Select Via form. ■ `select` or `nil`: Opens the Select Via form. ■ `up` or `Up`: Selects the default via for the layer above current layer ■ `down` or `Down`: Selects the default via for the layer below current layer or opens the Select Via form if the layers below current layer are implant layers. ■ `cancel`: Cancels any pending vias. ■ *layerName*: Closes the form and selects a via to connect to the specified layer. ■ *viaName*: Closes the form and selects the specified via.
g_rotate	Boolean which specifies whether to rotate the via or not. Default is `nil`.

Value Returned

`t`	A via is placed or the form is opened for via selection depending on the arguments.
`nil`	The window ID is invalid or specified via or layer can not be found.

Examples

```
leHiAddWireVia(hiGetCurrentWindow() "up")
```

Related Topics

Interactive and Assisted Routing Functions

Working With Vias

leSpaceBarFunc

leHiCancelStitch

```
leHiCancelStitch(
     [ w_windowId ]
     )
     => t / nil
```

Description

Cancels the via that is being added in a layout and returns to an active wire editing command, such as Create Wire or Create Bus.

Arguments

w_windowId	The current window ID.

Value Returned

t	The command completed successfully.
nil	The command did not completed successfully or an error occurred.

Examples

The following example cancels the via being added in the layout.

```
leHiCancelStitch()
```

The following example cancels the via being added in the specified window ID.

```
windowId = hiGetCurrentWindow()
leHiCancelStitch(windowId)
```

Related Topics

Interactive and Assisted Routing Functions

leHiCreateBus

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

Description

Starts the *Create – Bus* command in XL. You can press F3 to open the *Create Bus* form. The command lets you create buses.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was not successful or an error occurred.

Examples

The following example starts the *Create Bus* command and let you create buses.

```
leHiCreateBus()
```

Related Topics

Interactive and Assisted Routing Functions

Working with Buses

leHiCreateGeometricWire

```
leHiCreateGeometricWire(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Create – Shape – Geometric Wire* command. You can press F3 to open the Create Geometric Wire form. Using this command, you can create geometric wires comprising of paths or pathSegs and vias outside of routes. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	The current window ID.

Value Returned

t	The command was successful.
nil	The command was not successful or an error occurred.

Examples

The following example starts the *Create Geometric Wire* command and lets you create geometric wires.

```
leHiCreateGeometricWire()
```

Related Topics

Interactive and Assisted Routing Functions

Create Geometric Wire

leHiCreateStrandedWire

```
leHiCreateStrandedWire(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Create – Stranded Wire* command. You can press `F3` to open the Create Stranded Wire form. The command lets you create a wire comprising of multiple strands in a single direction on a single net.

Arguments

w_windowId	The current window ID. If *w_windowId* is not specified, the current window is used.

Value Returned

`t`	The command was successful.
`nil`	The command was not successful or an error occurred.

Examples

The following example starts the *Create Stranded Wire* command and lets you create a wire comprising of multiple strands in a single direction on a single net.

```
leHiCreateStrandedWire()
```

Related Topics

Interactive and Assisted Routing Functions

Working with Stranded Wires

leHiCreateWire

```
leHiCreateWire(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Create – Wire* command in XL. You can press F3 to open the *Create Wire* form. The command lets you create symbolic wires comprising of pathSegs and vias in routes. If *w_windowId* is not specified, the current window is used.

Arguments

w_windowId	The current window ID.

Value Returned

t	The command was successful.
nil	The command failed to run or an error occurred.

Examples

The following example starts the *Create Wire* command.

```
leHiCreateWire()
```

Related Topics

Interactive and Assisted Routing Functions

Creating Wires

leHiP2P

```
leHiP2P(
     [ w_windowId ]
     )
     => t / nil
```

Description

Invokes the interactive Point to Point routing command. The Point to Point command is supported only in Layout XL and higher tiers.

Arguments

w_windowId	ID of the window in which to invoke the point to point command.

Value Returned

`t`	The command was successful.
`nil`	The command failed to run or was canceled.

Examples

The following example starts the interactive Point to Point routing command.

```
leHiP2P()
```

Related Topics

Interactive and Assisted Routing Functions

Point to Point Routing

leHiStitchToLayer

```
leHiStitchToLayer(
     t_layerName
     [ g_rotate ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Inserts a via between the current layer paths segment and the layer name provided in the parameter when the Create Wire or Create Bus command is active.

Arguments

t_layername	A text string specifying any valid layer name. Example: "Metal 1".
g_rotate	Boolean which specifies whether to rotate the via or not. Default is nil.
w_windowId	The current window ID.

Value Returned

t	The command completed successfully.
nil	The command failed to run or an error occurred.

Examples

The following example inserts a via between the current layer paths segment and Metal2.

```
leHiStitchToLayer("Metal2")
windowId = hiGetCurrentWindow()
rotate = nil
```

Related Topics

Interactive and Assisted Routing Functions

leWECycleControlWire

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

Description

Allows cycling through the extreme and middle control wires in the selected bus during interactive bus editing. This functions runs successfully only when the *Create Wire* command is active. This function is available only in Layout XL and higher tiers.

You can also cycle through the control wires by using the bindkey `Control` + `Shift` + `Z`.

Arguments

None

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run or an error occurred.

Examples

The following example lets you cycle between the extreme and middle control wires in the selected bus during interactive bus editing.

```
leWECycleControlWire()
```

Related Topics

Interactive and Assisted Routing Functions

Using the Control Wire

leWECycleSnap

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

Description

Allows cycling of highlight during the *Create Wire* command through level 1 or top-level pin edges available within the *Aperture* distance from the cursor. The highlighted edge represents the starting point of wire creation.

Arguments

None

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run or was cancelled.

Examples

The following example lets you cycle the highlight between level 1 or top-level pin edges available within the *Aperture* distance from the cursor.

```
leWECycleSnap()
```

Related Topics

Interactive and Assisted Routing Functions

Smart Snapping in Interactive and Assisted Routing Commands

leWENoSnap

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

Description

Resets the point of starting wire creation to the current location of the cursor during the *Create Wire* command.

Arguments

None

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run.

Examples

The following example resets the point of wire creation to the current location.

```
leWENoSnap()
```

Related Topics

Interactive and Assisted Routing Functions

Smart Snapping in Interactive and Assisted Routing Commands

weAddCustomTransitionMenuItem

```
weAddCustomTransitionMenuItem(
     t_name
     t_customTransition
     [ t_bindkey ]
     )
     => t / nil
```

Description

Adds a menu item to the context-sensitive menu of the *Create Bus* command. You can use the added menu item to call the customer SKILL API. A bindkey is also assigned to the menu item.

Arguments

t_name	Name of the menu item in the *Create Bus* context-sensitive menu.
t_customTransition	Custom SKILL API to be called when the new menu item is clicked.
t_bindkey	Bindkey assigned to the menu item callback.

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run or an error occurred.

Examples

The following example adds a new menu item in the Multi-Layer bus submenu. The name of the menu item is `Via To M1/M2/M1/M2/.....` The `patternM1M2` is the name of a SKILL procedure that the customer has to define. This SKILL procedure describes how the tool has to behave when the RMB `Via To M1/M2/M1/M2/...` is clicked.

```
procedure (myCustomTransition(patternM1M2 cellViewId netLayerPairs)
   (let()
      list("Metal1" Metal2)
   )
)
```

```
weAddCustomTransitionMenuItem("Via To M1/M2/M1/M2/..." "patternM1M2")
```

Related Topics

Interactive and Assisted Routing Functions

weAutoTwigCycleTrunkViaAlignment

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

Description

Cycles through the different modes of via alignment depending on the via direction mode defined by the weAutoTwigTrunkViaAlignment environment variable.

Arguments

None

Value Returned

`t`	The twig or trunk via alignment has been cycled.
`nil`	The command failed to run or an error occurred.

Examples

The following example displays the default via alignment mode and the toggling between the alignment modes when the SKILL function is run.

```
envGetVal("we" "weAutoTwigTrunkViaAlignment" 'cyclic)
=>"center"

weAutoTwigCycleTrunkViaAlignment()

envGetVal("we" "weAutoTwigTrunkViaAlignment" 'cyclic)
=>"rightOrTop"
```

Related Topics

Interactive and Assisted Routing Functions

weAutoTwigTrunkViaDirection

weCWHoldWidth

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

Description

This function is used for the create wire command when smart snapping is active. The width that is highlighted by smart snapping is used by the next wire that is being created even if the wire is created at another position than the current one.

The default bindkey that can be used for this function is *Ctrl+I* and is used interactively.

Arguments

None

Value Returned

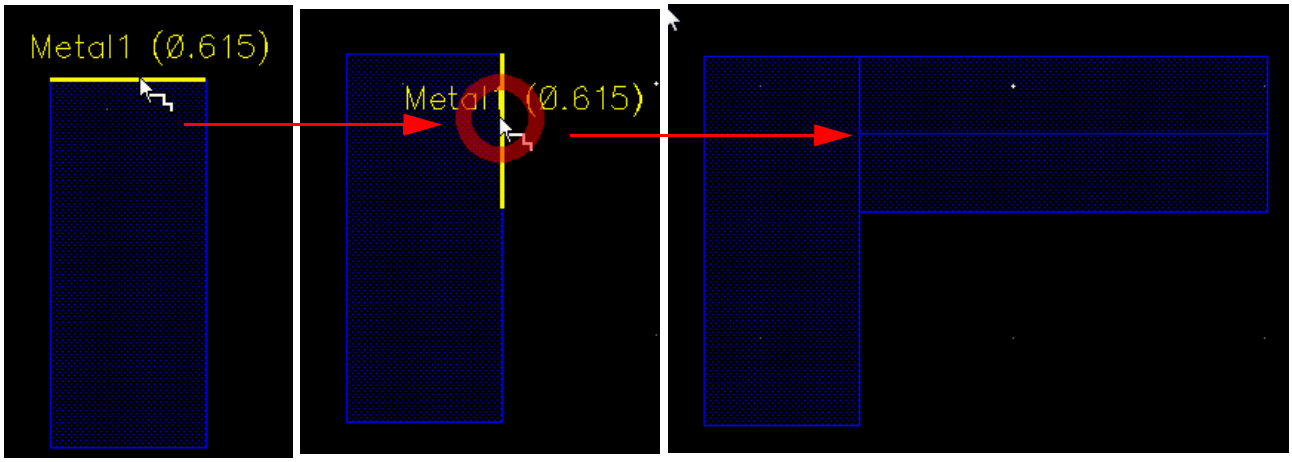
`t`	The command ran successfully.
`nil`	The command failed to run or an error occurred.

Examples

The following example creates a wire with the width that is highlighted by smart snapping.

```
weCWHoldWidth()
```

The following figure shows a wire of width 0.615 being created.



Related Topics

Interactive and Assisted Routing Functions

weCycleCutColorVia

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

Description

Cycles color of each cut when a via is inserted.

This SKILL API has no effect on gray mask color.

Arguments

None

Value Returned

`t`	The cut color pattern has been cycled.
`nil`	The command failed to run or an error occurred.

Examples

When you start the *Create Wire*, *Create Bus*, or *Create Stranded Wire* command, the `weCycleCutColor()` SKILL function cycles color of each via cut. For example, `mask1Color` to `mask2Color` to `mask3Color` and then back to `mask1Color`.

```
weCycleCutColor()
```

Related Topics

Interactive and Assisted Routing Functions

weCyclePatternGravityLayerTransitionType

```
weCyclePatternGravityLayerTransitionType(
     )
     => t_patternGravityLayerTransitionType / nil
```

Description

In a WSP design, the wires are snapped to the layer up or down WSPs, which is the default behavior. However, if you want the wire only to snap to layer up or layer down a WSP, then you can use the `patternGravityLayerTransitionType` environment variable to cycle the layer for snapping. The environment variable has three values: `patternGravityLayerUp`, `patternGravityLayerDown`, and `patternGravityLayerAny`. The default value is `patternGravityLayerAny`.

Arguments

None

Value Returned

`t_patternGravityLayerTransitionType`	The current value of the `patternGravityLayerTransitionType` environment variable.
`nil`	The command failed to run or an error occurred.

Examples

The following example, cycles and returns the current value of the `patternGravityLayerTransitionType` environment variable. For example, if current `patternGravityLayerTransitionType` is `patternGravityLayerAny`, the value is changed to the `patternGravityLayerUp` when this SKILL function is used.

```
weCyclePatternGravityLayerTransitionType()
"patternGravityLayerUp"
```

Related Topics

Interactive and Assisted Routing Functions

weGatherBusWires

```
weGatherBusWires(
     [ w_windowId ]
     )
     => t / nil
```

Description

This function is used when the `leHiCreateBus` command is running. If the gap between the bus wires does not match the *minSpacing* rule, then the wires are gathered at the minimum spacing. If the wires are already at `minSpacing`, they will be spread to use the initial gap, which is the gap before executing the `weGatherBusWires` function.

Arguments

w_windowId	ID of the window in which the `leHiCreateBus` command is running. By default it uses the current window.

Value Returned

`t`	The gap spacing of wires has been modified.
`nil`	The command failed to run or an error occurred.

Examples

The following example minimizes the spacing between bus bit to reach `minspacing` value.

```
weGatherBusWires(hiGetCurrentWindow())
```

Related Topics

Interactive and Assisted Routing Functions

weGetAdjustedWidthForTracks

```
weGetAdjustedWidthForTracks(
     d_cellViewId
     l_layerNum
     l_purposeNum
     l_direction
     l_dimension
     l_coordinates [ x_coord y_coord ]
     )
     => l_trackInfo / nil
```

Description

Returns the Width Spacing Pattern (WSP) track information based on the inputs specified, such as layers, purposes, preferred directions, and shape dimensions, at a specific location.The purpose of the function is to check whether or not there are tracks at the given location.

Arguments

d_cellViewId	ID of the currently edited cellview.
l_layerNum	List of layers in the design.
l_purposeNum	List of layer-purpose pairs in the design.
l_direction	List of preferred directions.
l_dimension	List of shapes dimension. The shapes dimension includes the width and height of the shape.
l_coordinates	Coordinates of the point of interest.

Value Returned

l_trackInfo	A list of lists representing the track information. The track information consists of the width, direction, `trackHasWidth`, and color, which is indexed in the list of lists returned by the SKILL function. Here is an example of the return value.
	`list(`
	`list(width1 width2 ... widthN) ; list of widths`
	`list(direction1 direction2 ... directionN) ; list of directions`
	`list(trackHasWidth1 trackHasWidth2 ... trackHasWidthN) ; list of boolean values indicating whether or not the track has a width`
	`list(maskColor1 maskColor2 ... maskColorN) ; list of colors`
	`)`

`nil`	The command fails to run or an error occurred.

Examples

Given below are two examples to explain how the SKILL function is used.

Example 1

The following example shows how to get the track information for a single layer/purpose/direction/shape's dimensions.

```
tech = techGetTechFile(dbGetCellView())
layer = techGetLayerNum(tech "M1")
purpose = techGetPurposeNum(tech "drawing")
weGetAdjustedWidthForTracks(dbGetCellView() list(layer) list(purpose)
list("horizontal") list(list(0.1 0.1)) list(0.3880 0.1240))
```

The value returned for the above example is:

```
(
  (0.032) - track width
  ("horizontal") - track direction
  (t) - Boolean value of trackHasWidth
  ("grayColor") - track color
)
```

At the location (`0.3880 0.1240`), for the `horizontal` preferred direction, considering the layer-purpose pair (`M1`, `drawing`), for a shape of dimension (`width= 0.1` and `height = 0.1`), there is a corresponding `M1` `horizontal` track with width as `0.032` and no color.

Example 2

The following example shows how to get the track information for multiple layers/purposes/ directions/shape's dimensions.

```
tech = techGetTechFile(deGetCellView())
layer1 = techGetLayerNum(tech "M1")
layer2 = techGetLayerNum(tech "M2")
layer3 = techGetLayerNum(tech "M3")
purpose = techGetPurposeNum(tech "drawing")
weGetAdjustedWidthForTracks(deGetCellView() list(layer1 layer2 layer3)
list(purpose purpose purpose) list("vertical" "vertical" "horizontal")
list(list(0.1 0.064) list(0.1 0.064) list(0.1 0.064)) list(0.3880 0.2040))
```

The value returned for the above example is:

```
(
  (0.048 0.0 0.064) - track widths
  ("vertical" "vertical" "horizontal") - direction of tracks
  (t nil t) - boolean value of trackHasWidth
  ("mask1Color" "grayColor" "mask2Color") - Color of tracks
)
```

From the given inputs, there are three matching tracks with the following characteristics:

- A `vertical` `M1` track with width as `0.048` and `mask1Color`.
- A `vertical` `M2` track with no width and no color.
- An `horizontal` `M3` track with width as `0.064` and `mask2Color`.

Related Topics

Interactive and Assisted Routing Functions

weGetCustomTransitionMenuItems

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

Description

Returns a list of custom transitions that were added to the *Create Bus* context-sensitive menu.

Arguments

None

Value Returned

`t`	Lists the available custom transitions in the *Create Bus* context-sensitive menu.
`nil`	The command failed to run or an error occurred.

Examples

The following example returns a list of the menu items that were added to the `Create Bus` context-sensitive menu using the `weAddCustomTransitionMenuItem` SKILL function.

```
procedure(myCustomTransition patternM1M2 cellViewId netLayerPairs)
   (let()
      list("Metal1" Metal2)
   )
)

weGetCustomTransitionMenuItems()
nil

weAddCustomTransitionMenuItem("Via To M1/M2/M1/M2/..." "patternM1M2")
weGetCustomTransitionMenuItems()
(("Via To M1/M2/M1/M2/..." "patternM1M2"))
```

Related Topics

Interactive and Assisted Routing Functions

weGetPathSegWidth

```
weGetPathSegWidth(
     [ g_pathSegId ]
     )
     => float
```

Description

Automatically handles the sqrt(2) conversion for diagonal path segments when their width is being queried. Unlike `dbGetPathSegWidth` which returns the value stored in the OpenAccess database for the width of diagonal pathSegs, this API does the sqrt(2) conversion appropriately and return the width value as it appears in the property editor form or object inspector assistant.

Arguments

g_pathSegID	OpenAccess database Id of the pathSeg being queried.

Value Returned

`float`	Returns an sqrt(2) converted width for diagonal pathSeg. For manhattan pathSegs, it returns the value of `debGetPathSegWidth` using `~>` operator.

Examples

Consider the following examples:

Consider ortho is a manhattan pathseg whose width == 0.3

```
\i weGetPathSegWidth(ortho)
\t 0.3
\p >
\i ortho~>width
\t 0.3
```

Consider diag is a diagonal pathseg whose width == 0.3. The following script shows the use of `weGetPathSegWidth` versus using `~>` operator with `dbGetPathSegWidth`. Here the diagonal width is calculated as 0.43 = mfgGrid (0.3 * sqrt(2))

```
\i weGetPathSegWidth(diag)
\t 0.3
```

```
\p >
\i diag~>width
\t 0.43
\p >
```

Related Topics

Interactive and Assisted Routing Functions

weHiCopyRoute

```
weHiCcopyRoute(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Edit – Wiring – Copy Route* command. Press `F3` in the canvas to open the Copy Route form. The command lets you copy layout geometries similar to a bus from a single routed net.

Arguments

w_windowId	The window ID in which the command is to be run. If the window ID is not specified, current window ID is used.

Value Returned

`t`	The command was successful.
`nil`	The command was not successful or an error occurred.

Examples

The following example starts the *Copy Route* command for the design in the currently active window(?) and lets you create layout geometries similar to a bus.

```
weHiCopyRoute()
```

Related Topics

Interactive and Assisted Routing Functions

weHiCycleViaDefDown

```
weHiCycleViaDefDown(
     )
     => t_viaDefName / nil
```

Description

Returns the current via definition for the pending vias when you start either the *Create Wire*, *Create Stranded Wire*, or the *Create Bus* command.

Arguments

None

Value Returned

`t_viaDefName`	The new via definition for the pending vias.
`nil`	The command failed to run or an error occurred.

Examples

When either *Create Wire*, *Create Bus*, or *Create Stranded Wire* command is started and vias are pending, the SKILL function cycles down the list of available via definitions for each via, applies the new via definition to each pending via, and returns the new value.

```
weHiCycleViaDefDown()
"M1_PSUB"
```

Related Topics

Interactive and Assisted Routing Functions

weHiCycleViaDefUp

```
weHiCycleViaDefUp(
     )
     => t_viaDefName / nil
```

Description

Automatically cycles up the via definition for the pending vias, while using the *Create Wire*, *Create Stranded Wire*, or the *Create Bus* command.

Arguments

None

Value Returned

`t_viaDefName`	The new via definition for the pending vias.
`nil`	The command failed to run or an error occurred.

Examples

When either *Create Wire*, *Create Bus*, or *Create Stranded Wire* command is started and vias are pending, the SKILL function cycles down the list of available via definitions for each via, applies the new via definition to each pending via, and returns the new value.

```
weHiCycleViaDefUp()
"M1_NWELL"
```

Related Topics

Interactive and Assisted Routing Functions

weHiEditBus

```
weHiEditBus(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the *Edit – Wiring – Bus* command in Layout XL and higher tiers. The command lets you edit the width and spacing of bus bits. If no window ID is specified, the current window is used.

Arguments

w_windowId ID of the window where the Edit Bus command is to be invoked.

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run or an error occurred.

Examples

The following example starts the Edit Bus command.

```
weHiEditBus()
```

Related Topics

Interactive and Assisted Routing Functions

Editing a Bus

weHiInteractiveRouting

```
weHiInteractiveRouting(
     [ w_windowId ]
     )
     => t / nil
```

Description

Maps the function to the leHiCreateGeometricWire (paths mode) SKILL function when the technology file is one of the fabric types, package, module, or board, and the active application is Virtuoso RF. Otherwise, the function is mapped to leHiCreateWire when the technology file is of the fabric type IC or the active application is not Virtuoso RF.

Arguments

w_windowId	ID of the window where the interactive routing command is to be invoked.

Value Returned

t	The command ran successfully.
nil	The command failed to run or an error occurred.

Examples

The following example specifies the ID of the current window where the interactive routing command is to be invoked.

```
weHiInteractiveRouting(hiGetCurrentWindow())
```

Related Topics

Interactive and Assisted Routing Functions

weHiWireTo45

```
weHiWireTo45(
     [ w_windowId ]
     )
     => t / nil
```

Description

Starts the interactive command that lets you select wire corners to create chamfering.

Arguments

w_windowId	ID of the window where the interactive command is to be started. If *w_windowId* is not specified, the current window is used

Value Returned

t	The command ran successfully.
nil	The command failed to run or an error occurred.

Examples

```
weHiWireTo45()
```

Related Topics

Interactive and Assisted Routing Functions

weRemoveCustomTransitionMenuItem

```
weRemoveCustomTransitionMenuItem(
     t_name
     )
     => t / nil
```

Description

Removes a custom transition menu item from the *Create Bus* context-sensitive menu.

Arguments

t_name	Name of the menu item to be removed from the *Create Bus* context-sensitive menu.

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run or an error occurred.

Examples

The following example removes the name of the menu item `Via To M1/M2/M1/M2/...` from the *Create Bus* context-sensitive menu.

```
procedure(myCustomTransition patternM1M2 cellViewId netLayerPairs)
   (let()
      list("Metal1" Metal2)
   )
weRemoveCustomTransitionMenuItem("Via To M1/M2/M1/M2/...")
weGetCustomTransitionMenuItems()
nil
```

Related Topics

Interactive and Assisted Routing Functions

weScaleMagnifierOrDecreaseWidth

```
weScaleMagnifierOrDecreaseWidth(
     [ w_windowId ]
     )
     => t / nil
```

Description

If the magnifier is opened, it performs a zoom out in the magnifier. Else if *Create Bus*, *Create Wire*, or *Reshape* command is being executed, decrease the width of the current wire.

Arguments

w_windowId	The current window ID.

Value Returned

t	The command ran successfully.
nil	The command failed to run or an error occurred.

Examples

The following example reduces the width of the edited segment.

```
weScaleMagnifierOrDecreaseWidth(hiGetCurrentWindow())
```

Related Topics

Interactive and Assisted Routing Functions

weScaleMagnifierOrIncreaseWidth

```
weScaleMagnifierOrIncreaseWidth(
     [ w_windowId ]
     )
     => t / nil
```

Description

If the magnifier is opened, it performs a zoom in the magnifier. Else if Create Bus, Create Wire, or Reshape command is being executed, increase the width of the current wire.

Arguments

w_windowId	The current window ID.

Value Returned

t	The command ran successfully.
nil	The command failed to run or an error occurred.

Examples

The following example increases the width of the edited segment.

```
weScaleMagnifierOrIncreaseWidth(hiGetCurrentWindow())
```

Related Topics

Interactive and Assisted Routing Functions

weScrollOrCycleDownWireViaAlignment

```
weScrollOrCycleDownWireViaAlignment(
     [ w_windowId ]
     )
     => t / nil
```

Description

Lets you cycle through the wire and via alignment settings when a via is being dragged in the layout canvas. Else, the `weScrollOrCycleDownWireViaAlignment` SKILL API scroll down the layout canvas by pressing the `Ctrl` key and clicking the mouse wheel. This function works only in the *Create Bus*, *Create Wire, and Reshape* commands.

Arguments

`w_windowId` ID of the window where the command is to be invoked.

Value Returned

`t`	The command ran successfully.
`nil`	The command fails to run or an error occurred.

Examples

The following example allows the via alignment to cycle down from the bottom right value to the top left value.

```
weScrollOrCycleDownWireViaAlignment(windowId)
```

Related Topics

Interactive and Assisted Routing Functions

weScrollOrCycleUpWireViaAlignment

```
weScrollOrCycleUpWireViaAlignment(
     [ w_windowId ]
     )
     => t / nil
```

Description

Lets you cycle through the wire and via alignment settings when a via is being dragged in the layout canvas. Else, the `weScrollOrCycleUpWireViaAlignment` SKILL API scrolls up the layout canvas by pressing the `Ctrl` key and clicking the mouse wheel. This function works only in the *Create Bus*, *Create Wire, and Reshape* commands.

Arguments

`w_windowId` ID of the window where the command is to be invoked.

Value Returned

`t`	The command ran successfully.
`nil`	The command fails to run or an error occurred.

Examples

The following example allows the via alignment to cycle up from the top left value to the bottom right value.

```
weScrollOrCycleUpWireViaAlignment(windowId)
```

Related Topics

Interactive and Assisted Routing Functions

weSetPathSegWidth

```
weSetPathSegWidth(
     [ g_pathSegId ]
     float value
     )
     => t / nil
```

Description

Automatically handles the sqrt(2) conversion for diagonal pathseg when the width is being changed. In addition, it also supports diagonal pathseg with custom extension and appropriately updates the values when the width is being changed.

Arguments

g_pathSegID	OpenAccess database Id of the pathSeg being queried.
float value	The new width value that is to be set for the pathseg.

Value Returned

t	The command ran successfully.
nil	The command failed to run or an error occurred.

Examples

Consider that diag is a diagonal pathseg whose width == 0.3 and ortho is a diagonal pathseg whose width == 0.3. The following script shows the use of `dbSetPathSegWidth`.

```
\i weSetPathSegWidth(diag 0.5)
\t t
\p >
\i weGetPathSegWidth(diag)
\t 0.5
\p >
\i diag~>width
\t 0.71
\p >
```

Related Topics

Interactive and Assisted Routing Functions

13

Slot Functions

This topic provides a list of Cadence® SKILL functions associated with slotting.

The SKILL functions documentation provides syntax, descriptions, and examples for the Cadence® SKILL functions associated with slotting.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- sltSetContextSpecificationCells
- sltShapeConsecutiveSlotting

sltSetContextSpecificationCells

```
sltSetContextSpecificationCells(
     l_contextSpecificationCells
     )
     => t / nil
```

Description

Defines a list of contexts for slotting. A slotting context has a name and an associated slotting specification cell in the format `library/cell/view`.

Arguments

`l_contextSpecificationCells`

List of context pairs defined for slotting.

Value Returned

`t`	A list of contexts was defined for slotting.
`nil`	The command failed to run.

Examples

Adding the following example in the `.cdsinit` file defines a list of contexts for slotting. The two environment variables that let you use these defined contexts are `useOneSlotSpecCell` and `slottingContext`.

```
sltSetContextSpecificationCells( '(("context1" "DESIGN/specCell1/layout")
("context2" "DESIGN/specCell2/layout")) )
```

After specifying the two environment variables, when you open the Metal Density Options form, the slotting parameters are taken from the specification cell `DESIGN/specCell2/layout`.

```
envSetVal("slt" "useOneSlotSpecCell" 'boolean nil)
envSetVal("slt" "slottingContext" 'string "context2")
```

Related Topics

Slot Functions

sltShapeConsecutiveSlotting

`lxShapeSlotting`

`lxConvertSlotToPolygon`

sltShapeConsecutiveSlotting

```
sltShapeConsecutiveSlotting(
     [ ?cv d_cellViewId ]
     [ ?all { t | nil } ]
     [ ?region l_region ]
     [ ?layers l_layers ]
     [ ?objects l_objects ]
     [ ?slotLength n_slotLength ]
     [ ?slotWidth n_slotWidth ]
     [ ?slotStaggered { t | nil } ]
     [ ?lengthWidthRatio n_lengthWidthRatio ]
     [ ?slotVia { t | nil } ]
     [ ?excludePins { t | nil } ]
     [ ?widthThreshold1 n_widthThreshold1 ]
     [ ?widthThreshold2 n_widthThreshold2 ]
     [ ?lSpacing1 n_lSpacing1 ]
     [ ?lSpacing2 n_lSpacing2 ]
     [ ?wSpacing1 n_wSpacing1 ]
     [ ?wSpacing2 n_wSpacing2 ]
     [ ?slotToEdge1 n_slotToEdge1 ]
     [ ?slotToEdge2 n_slotToEdge2 ]
     [ ?slotSpacingAtTurn1 n_slotSpacingAtTurn1 ]
     [ ?slotSpacingAtTurn2 n_slotSpacingAtTurn2 ]
     )
     => t / nil
```

Description

Consecutively runs density-aware slotting and geometric slotting with a combination of common and specific parameters.

This SKILL API invokes the `lxShapeSlotting` SKILL procedure twice. The first run is in the density-aware mode and the second run is in geometric mode. The parameters of the SKILL API are directly passed to the `lxShapeSlotting` SKILL procedure for the corresponding parameters. Some parameters are passed to both runs and some are passed in one run only (their name having 1 or 2 as suffix).

- The common parameters are:
  - `cv`
  - `all`
  - `layers`
  - `region`
  - `objects`

- `slotLength`
- `slotWidth`
- `lengthWidthRatio`
- `slotStaggered`
- `slotVia`
- `excludePins`

- The density-aware mode parameters are:
  - `slotToEdge1`
  - `lSpacing1`
  - `wSpacing1`
  - `widthThreshold1`
  - `slotSpacingAtTurn1`
- The geometric mode parameters are:
  - `slotToEdge2`
  - `lSpacing2`
  - `wSpacing2`
  - `widthThreshold2`
  - `slotSpacingAtTurn2`

Arguments

`?cv`	ID of the cellview containing the layout instance. If not specified, the current cellview is used.
`?all`	Applies the command to the entire design when set to `t`. Default is `t`.
`?region`	Applies the command to the region given by a list of coordinates (x0 y0 x1 y1) representing the lower-left and upper-right corners of the region.

`?layers`	List of layer names on which the `drawing` purpose metal shapes will be overlapped by `slot` purpose shapes. The default is all layers.
`?objects`	List of objects to be slotted. The object can be a path segment, path, rectangle, or polygon. If `?region` is also specified, then only the section of the objects that are partially or fully covered by the `?region` are slotted.
`?widthThreshold1` `?widthThreshold2`	Filters out the shapes with widths smaller than the one specified. Default is `12.0µm`.
`?slotLength`	Defines the length of the slot. Default is `0.0µm`. If the specified value is negative, then the value of the `slotLength` environment variable is used instead.
`?slotWidth`	Defines the width of the slot. Default is `0.0µm`. If the specified value is negative, then the value of the `slotWidth` environment variable is used instead.
`?slotStaggered`	Defines the offset mode used to add the slots. When set to `nil`, the slots are in line. When set to `t`, slots are staggered. By default the argument is set to `nil`.
`?lengthWidthRatio`	Specifies a threshold ratio value, which determines whether a slot is created as a square or a rectangular shape. If the length-to-width ratio of the object is greater than the defined threshold, then a rectangular slot is created. If the length-to-width ratio of the object is less than or equal to the specified ratio, then a square slot is created. The default is `3.0`. If the specified value is negative, then the value of the `lengthWidthRatio` environment variable is used instead.
`?slotVia`	Specifies that vias are to be slotted. Default is `nil`.
`?excludePins`	Excludes pin objects from being slotted. Default is `nil`.
`?lSpacing1` `?lSpacing2`	Defines the slot spacing in the length direction. Default is `0.0µm`. If the specified value is negative, then the value of the `lSpacing` environment variable is used instead.

`?wSpacing1`
`?wSpacing2`

Defines the slot spacing in the width direction. Default is `0.0µm`.

If the specified value is negative, then the value of the `wSpacing` environment variable is used instead.

`?slotToEdge1`
`?slotToEdge2`

Defines the minimum slot to edge distance. Default is `0.0µm`.

If a value is negative then the value of `slotToEdge` environment variable is used instead.

`?slotSpacingAtTurn1`
`?slotSpacingAtTurn2`

Specifies the spacing that needs to be applied at the turns or on vias.

If this argument is not specified:

- In density-aware mode, the command computes the spacing between the slots in the turn from the target density.
- In geometric mode, the slot shapes at turns are aligned with the slot shapes from the wires.

When the argument is specified:

- If the value = 0, then the slots in the turn are aligned with the slots of the shapes touching the turn.
- If the value > 0, then the value is used as the spacing between the slots in the turn.

Value Returned

`t`	The command ran successfully.
`nil`	The command failed to run due to invalid input parameters.

Examples

The following lines of code in the CIW runs `lxShapeSlotting` consecutively.

```
slotLength      = 6.0
slotWidth       = 1.0
slotToEdge      = 1.0
```

```
    lSpacing         = 2.0
    widthThreshold   = 12.0
    lengthWidthRatio = 0.5
    slotVia          = t
    wSpacing          = 5.2
    slotSpacingAtTurn = 2.0

sltShapeConsecutiveSlotting(
    ?cv                 cvId
    ?all                nil
    ?layers             extendedSlotLayers
    ;?region
    ?objects            objects
    ?slotLength         slotLength
    ?slotWidth          slotWidth
    ?lengthWidthRatio   lengthWidthRatio
    ?slotStaggered      nil
    ?excludePins        t
    ?slotVia            slotVia
    ?slotToEdge2        slotToEdge
    ?lSpacing1          lSpacing
    ?lSpacing2          lSpacing
    ;?wSpacing1
    ?wSpacing2          wSpacing
    ?widthThreshold1    widthThreshold
    ?widthThreshold2    widthThreshold
    ;?slotSpacingAtTurn1 -0.1   (let density aware
    ?slotSpacingAtTurn2 slotSpacingAtTurn)
```

Related Topics

Slot Functions

sltShapeConsecutiveSlotting

lxShapeSlotting

lxConvertSlotToPolygon

14

Design Rule Driven Editing Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso® Layout Suite Design Rule Driven (DRD) editing feature.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- drdAddTarget
- drdBatchCheck
- drdBatchCheckLicenseAvailable
- drdCheckedConstraintGroupItem
- drdCheckedLayerGroupItem
- drdCheckedRuleGroupItem
- drdConstraintGroupHeaderClicked
- drdConstraintHeaderClicked
- drdConstraintItemClicked
- drdEnablePixelThreshold
- drdGetAllowedWidth
- drdGetMinSpacing
- drdGetMinSpanLengthSpacing
- drdGetMinVoltageSpacing
- drdIsPixelThresholdEnabled
- drdLayerGroupHeaderClicked
- drdLayerHeaderClicked

- drdLayerItemClicked
- drdListConstraintCategoryRules
- drdOptionsSet
- drdOptionUpdateConstraint
- drdOptionUpdateLayer
- drdRemoveTarget
- drdRuleGroupHeaderClicked
- drdRuleIDHeaderClicked
- drdRuleIDItemClicked
- drdRuleTypeHeaderClicked
- drdRuleTypeItemClicked
- drdSelectConstraintGroupItem
- drdSelectLayerGroupItem
- drdSelectSignOffRuleGroupItem
- drdToggleSmartSnapMode
- drdToggleSmartSnapModeForDiscreteSpacing
- drdVerifySelSet
- leHiBatchChecker
- leToggleDrdMode

drdAddTarget

```
drdAddTarget(
     w_winID
     l_point
     )
     => t / nil
```

Description

Sets the shape located at the specified point in the specified window as a DRD target. The function does not remove from the selection the shapes already set as DRD targets. If two or more shapes overlap at the specified point, the shape on the active layer is set as a DRD target.

For example, if a shape on `Poly` layer and a shape on `Metal1` layer overlap at the specified point and you want to set the shape on `Poly` layer as a DRD target, set `Poly` as the active layer in the Palette.

You can bind a key to this function and use the bindkey during interactive editing to set objects as DRD targets. For more information, see Setting DRD Targets Using the Bindkey and Bindkey Reassignment in *Virtuoso Design Rule Driven Editing User Guide*.

Arguments

w_winID	ID of the layout window in which you want to set DRD targets.
l_point	Coordinates of the location in the layout window where the target is to be set. The coordinates must be specified in database units (dbu).

Value Returned

`t`	A shape at the specified point was set as a DRD target.
`nil`	No shape was set as a DRD target.

Examples

```
drdAddTarget(hiGetCurrentWindow() 2000:1000)
drdAddTarget(window(2) 10000:10000)
drdAddTarget(window(3) list(12000 1000))
```

drdBatchCheck

```
drdBatchCheck(
     d_cellViewID
     [ l_bBox ]
     )
     => t / nil
```

Description

Runs DRD checker on the specified area of the specified cellview. The hierarchy depth and batch checking rules specified in the DRD Options form are honored. Use this function to run a DRD check on read-only cellviews to view any DRC violations as warning messages in CIW.

Arguments

d_cellViewID	ID of the cellview to be checked.
l_bBox	Area of the cellview to be checked. Specify a pair of coordinates to denote the lower-left and upper-right corners of the bounding box, such as: `drdBatchCheck(cv list(0:0 1:1))` If this argument is not specified, the entire cellview bounding box is considered by default. The bounding box is in user units.

Value Returned

`t`	The checker ran successfully.
`nil`	The checker failed.

Examples

```
cv = dbOpenCellViewByType("libName" "cellName" "viewName" "maskLayout" "a")
drdBatchCheck(cv); checks the whole cellview
drdBatchCheck(cv list(0:0 1:1)); checks the cellview box (0,0) to (1,1)
```

Shown below is a warning message displayed in CIW when `drdBatchCheck`() is run on a read-only cellview.

```
*WARNING* Read-only Marker: Tool="drdEdit", Ref="METAL2.W.1", Desc="Minimum Metal2
Width 0.08", Layers="Metal2", Purposes="oaNo",
Pts="(271.050000,25.445000)(271.050000,25.510000)(271.130000,25.510000)(271.13000
0,25.445000)"
```

drdBatchCheckLicenseAvailable

```
drdBatchCheckLicenseAvailable(
     [ x_timeToWaitInSeconds ]
     )
     => t / nil
```

Description

Checks whether a Layout XL license is available. If the license is available, the function returns `t`. If the license is unavailable, it attempts to check out the license. If the attempt is successful, it checks the license back in and returns `t`.

Arguments

x_timeToWaitInSeconds

Specifies the duration (in seconds) for which to wait for a Layout XL license to become available.

The default is `0`.

Value Returned

`t`	Layout XL license is available.
`nil`	Layout XL license is not available.

Example

```
gotLicense = drdBatchCheckLicenseAvailable(30)
```

drdCheckedConstraintGroupItem

```
drdCheckedConstraintGroupItem(
     t_CategoryName
     x_toggleValue
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of a constraint category in the *Constraint Category* column on the *Filters* tab of the DRD Options form.

Arguments

t_CategoryName	Name of the constraint category.
x_toggleValue	Value indicating the different selection states of a constraint category; `0` for deselection, `1` for partial selection, and `2` for complete selection.
w_windowId	The current window ID.

Value Returned

`t`	The state of the constraint category was changed.
`nil`	The state of the constraint category could not be changed.

Examples

```
drdCheckedConstraintGroupItem("Width" 0 window(2))
```

It deselects the Width category.

```
drdCheckedConstraintGroupItem("Length" 1 window(2))
```

It selects the Length category partially.

```
drdCheckedConstraintGroupItem("Area" 2 window(2))
```

It selects the Area category.

drdCheckedLayerGroupItem

```
drdCheckedLayerGroupItem(
     t_layerCategoryName
     x_toggleValue
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of a layer category in the *Layer Category* column on the *Filters* tab of the DRD Options form.

Arguments

t_layerCategoryName	Name of the layer category.
x_toggleValue	Value indicating the different selection states of a layer category; `0` for deselection, `1` for partial selection, and `2` for complete selection.
w_windowId	The current window ID.

Value Returned

`t`	The state of the layer category was changed.
`nil`	The state of the layer category could not be changed.

Examples

```
drdCheckedLayerGroupItem("Device" 2 window(2))
```

It selects the Device category.

```
drdCheckedLayerGroupItem("Routing" 0 window(2))
```

drdCheckedRuleGroupItem

```
drdCheckedRuleGroupItem(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of check boxes.

Arguments

x_columnIndex	The column index.
w_windowId	The current window ID.

Value Returned

t	The command was successful.
nil	The command was unsuccessful.

Examples

```
drdCheckedRuleGroupItem(1 window(2))
```

Toggles the selection of the second column.

drdConstraintGroupHeaderClicked

```
drdConstraintGroupHeaderClicked(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Changes the state of the check boxes for all the categories listed in the *Constraint Category* column on the *Filters* tab of the DRD Options form.

Arguments

x_columnIndex	Column index; `0` indicates the *Constraint Category* column, `1` indicates the column under which the check boxes for the constraint categories are displayed.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdConstraintGroupHeaderClicked(0 window(2))
```

It performs no action.

```
drdConstraintGroupHeaderClicked(1 window(2))
```

It toggles the state of the check boxes for all the constraint categories.

drdConstraintHeaderClicked

```
drdConstraintHeaderClicked(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Changes the state of the check boxes for all the constraints listed in the *Constraint* column or sorts the constraints alphabetically, on the *Filters* tab of the DRD Options form.

Arguments

x_columnIndex	Column index; `0` indicates the *Constraint* column, `1` indicates the Enforce (*E*) column, `2` indicates the Notify (*N*) column, `3` indicates the Post-Edit (*PE*) column, and 4 indicates the Batch (*B*) column.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdConstraintHeaderClicked(0 window(2))
```

It sorts the constraints alphabetically.

```
drdConstraintHeaderClicked(1 window(2))
```

It toggles the state of the check boxes in the *E* column.

```
drdConstraintHeaderClicked(2 window(2))
```

It toggles the state of the check boxes in the *N* column.

```
drdConstraintHeaderClicked(3 window(2))
```

It toggles the state of the check boxes in the *PE* column.

```
drdConstraintHeaderClicked(4 window(2))
```

It toggles the state of the check boxes in the *B* column.

drdConstraintItemClicked

```
drdConstraintItemClicked(
     t_constraintName
     x_columnIndex
     x_stateValue
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of a constraint for different checking modes, that is, Enforce, Notify, Post-Edit, and Batch, on the *Filters* tab of the DRD Options form.

Arguments

t_constraintName	Name of the constraint.
x_columnIndex	Column index; `0` indicates the *Constraint* column, `1` indicates the Enforce (*E*) column, `2` indicates the Notify (*N*) column, `3` indicates the Post-Edit (*PE*) column, and `4` indicates the Batch (*B*) column.
x_stateValue	Value indicating the selection states of a constraint; `0` for deselection, `1` for partial selection, and `2` for complete selection.
w_windowId	The current window ID.

Value Returned

`t`	The state of the category was changed.
`nil`	The state of the category could not be changed.

Examples

```
drdConstraintItemClicked("maxNumCorners" 2 2 window(2))
```

It enables the Notify checking mode for the `maxNumCorners` constraint.

```
drdConstraintItemClicked("maxNumCorners" 2 0 window(2))
```

It disables the Notify checking mode for the `maxNumCorners` constraint.

```
drdConstraintItemClicked("maxNumCorners" 3 0 window(2))
```

It disables the Post-Edit checking mode for the `maxNumCorners` constraint.

```
drdConstraintItemClicked("maxNumCorners" 4 2 window(2))
```

It enables the Batch checking mode for the `maxNumCorners` constraint.

drdEnablePixelThreshold

```
drdEnablePixelThreshold(
     g_enable
     )
     => t / nil
```

Description

Enables a pixel threshold. When you move an object toward another object and you cross this pixel threshold, the object being moved is automatically snapped to the closest edge of the DRC halo or the target object, if DRD Smart Snap mode is enabled. You can disable this pixel threshold to temporarily disable smart snapping during interactive editing.

Note: By default, this SKILL function is assigned to bindkey `1`.

Arguments

g_enable	When set to `t`, the pixel threshold is enabled. When set to `nil`, the pixel threshold is disabled. The default is `t`.

Value Returned

`t`	The pixel threshold was enabled.
`nil`	The pixel threshold was not enabled.

Examples

```
drdEnablePixelThreshold(t)
```

Enables the pixel threshold, which, in turn, enables snapping.

```
drdEnablePixelThreshold(nil)
```

Disables the pixel threshold.

drdGetAllowedWidth

```
drdGetAllowedWidth(
     d_objID
     t_layerName
     [ ?constraintGroup t_constrGroupName ]
     [ ?direction { any | horizontal | vertical } ]
     [ ?upperLimit f_maxValue ]
     )
     => l_integers / nil
```

Description

Returns a list of allowed widths for the specified layer.

Arguments

d_objID	Database ID of the selected object.
t_layerName	The layer name.
?constraintGroup *t_constrGroupName*	The constraint group in which drdGetAllowedWidth is defined. This is a user-defined setup constraint group. The default value is foundry.
?direction { any | horizontal | vertical }	The direction in which width is measured. Valid values: horizontal, vertical, any (default)
?upperLimit *f_maxValue*	Specifies the upper bound up to which width values are to be retrieved.

Value Returned

l_integers	A list of legal width values (integers) less than or equal to the value specified using ?upperLimit.
nil	The width is not defined for the specified layer.

Examples

```
objID = css()
drdGetAllowedWidth(objID "M2" ?constraintGroup "foundry" ?upperLimit 0.05 )
=> (32 37 42 47)
```

Returns width values less than 0.05 for layer `M2`.

```
drdGetAllowedWidth(objID "M2" ?constraintGroup "foundry" ?direction "vertical"
?upperLimit 0.1 )
=> ( 32 37 42 47 52
     57 62 67 72 77
     82 87 92 97
   )
```

Returns width values less than 0.1 for layer `M2` when the direction is specified as `vertical`.

drdGetMinSpacing

```
drdGetMinSpacing(
     d_techID
     t_layerName
     [ ?constraintGroup t_constrGroupName ]
     [ ?width1 f_width1Val ]
     [ ?width2 f_width2Val ]
     [ ?prl f_prlVal ]
     [ ?direction { any | horizontal | vertical } ]
     [ ?sameMask { 0 | 1 } ]
     )
     => f_minSpaceVal / nil
```

Description

Returns the minimum spacing value defined by the minSpacing (One layer) constraint for the specified layer, based on the specified width, parallel run length (prl), direction, and mask values.

For the optional width1, width2, and prl arguments:

- If the minSpacing constraint is a "scalar", that is, defined without a lookup table, do not specify width1, width2, and prl.
- If the minSpacing constraint is indexed on width, specify the larger of the two widths as width1.
- If the minSpacing constraint is indexed on width and length, specify the larger of the two widths as width1 and the prl between the shapes as prl.
- If the minSpacing constraint is indexed on twoWidths and length, specify the width of the first shape as width1, the width of the second shape as width2, and the prl between them as prl.

Arguments

d_techID	The database identifier of the technology library.
t_layerName	The layer name.
?constraintGroup *t_constrGroupName*	The constraint group in which minSpacing is defined. This is a user-defined setup constraint group. The default value is foundry.

`?width1` *`f_width1Val`*

The width of the first shape or the larger of the two shapes.

`?width2` *`f_width2Val`*

The width of the second shape.

`?prl` *`f_prlVal`*

The prl between the two shapes.

`?direction { any | horizontal | vertical }`

The direction in which spacing is measured.
Valid values: `horizontal`, `vertical`, `any` (default)

`?sameMask { 0 | 1 }`

Specifies whether the spacing is to be retrieved for shapes on the same mask or on different masks.
Valid values: `0` (different mask), `1` (same mask, default)

Value Returned

`f_minSpaceVal`	The minimum spacing value.
`nil`	The minimum spacing is not defined for the specified layer or for the specified arguments.

Examples

```
cv = geGetEditCellView()
tech = techGetTechFile(cv)
drdGetMinSpacing(tech "Metal5" ?sameMask 0)
=> 0.032
```

Returns the minimum spacing defined between two `Metal5` shapes on different masks.

```
drdGetMinSpacing(tech "Metal5" ?width1 0.032 ?prl -0.025)
=> 0.048
```

Returns the minimum spacing defined between two `Metal5` shapes on the same mask (default). The width of the larger of the two shapes is greater than or equal to 0.32 and the prl between them is greater than or equal to -0.025.

```
drdGetMinSpacing(tech "Metal5" ?width1 0.032 ?prl 0.3 ?sameMask 0)
=> 0.07
```

Returns the minimum spacing defined between two `Metal5` shapes on different masks. The width of the larger of the two shapes is greater than or equal to 0.32 and the prl between them is greater than or equal to 0.3.

drdGetMinSpanLengthSpacing

```
drdGetMinSpanLengthSpacing(
     d_techID
     t_layerName
     [ ?constraintGroup t_constrGroupName ]
     [ ?span f_spanVal ]
     [ ?prl f_prlVal ]
     [ ?direction { any | horizontal | vertical } ]
     [ ?sameMask { 0 | 1 } ]
     )
     => f_minSpanSpaceVal / nil
```

Description

Returns the minimum span length spacing value defined by the minSpanLengthSpacing constraint for the specified layer, based on the specified span, parallel run length (prl), direction, and mask values.

Arguments

d_techID	The database identifier of the technology library.
t_layerName	The layer name.
?constraintGroup *t_constrGroupName*	The constraint group in which minSpanLengthSpacing is defined. This is a user-defined setup constraint group. The default value is foundry.
?span *f_spanVal*	The larger of the two spans.
?prl *f_prlVal*	The prl between the two shapes.
?direction { any | horizontal | vertical }	The direction in which spacing is measured. Valid values: horizontal, vertical, any (default)
?sameMask { 0 | 1 }	Specifies whether the spacing is to be retrieved for shapes on the same mask or on different masks. Valid values: 0 (different mask), 1 (same mask, default)

Value Returned

f_minSpanSpaceVal	The minimum span length spacing value.
nil	The minimum span length spacing is not defined for the specified layer or for the specified arguments.

Examples

```
cv = geGetEditCellView()
tech = techGetTechFile(cv)
drdGetMinSpanLengthSpacing(tech "Metal1" ?span 0.215)
=> 0.14
```

Returns the minimum span length spacing defined between two Metal1 shapes. The span of the larger of the two shapes is greater than or equal to 0.215.

```
drdGetMinSpanLengthSpacing(tech "Metal1" ?span 0.395 ?prl 0.0)
=> 0.13
```

Returns the minimum span length spacing defined between two Metal1 shapes. The span of the larger of the two shapes is greater than or equal to 0.395 and the prl between the shapes is greater than or equal to 0.0.

```
drdGetMinSpanLengthSpacing(tech "Metal1" ?span 0.590 ?prl -0.1 ?direction
"vertical")
=> 0.12
```

Returns the minimum span length spacing defined between two Metal1 shapes. The span of the larger of the two shapes is greater than or equal to 0.590 and the prl between the shapes is greater than or equal to -0.1. The spacing direction is vertical.

```
drdGetMinSpanLengthSpacing(tech "Metal1" ?span 0.590 ?prl -0.1 ?direction
"horizontal")
=> nil
```

Returns nil because span length spacing in the horizontal direction is not defined.

```
drdGetMinSpanLengthSpacing(tech "Metal1" ?span 0.590 ?prl -0.1 ?direction
"vertical" ?sameMask 1)
=> 0.12
```

Returns the minimum span length spacing defined between two same-mask Metal1 shapes. The span of the larger of the two shapes is greater than or equal to 0.590 and the prl between the shapes is greater than or equal to -0.1. The spacing direction is vertical.

drdGetMinVoltageSpacing

```
drdGetMinVoltageSpacing(
     d_techID
     t_layerName
     [ ?constraintGroup t_constrGroupName ]
     [ ?voltage1 f_voltage1Val ]
     [ ?voltage2 f_voltage2Val ]
     )
     => f_minVolSpaceVal / nil
```

Description

Returns the minimum voltage-dependent spacing value defined by the minVoltageSpacing (One layer) constraint for the specified layer based on the specified voltages. The voltage difference is calculated as the absolute difference of the two input voltages, which correspond to the voltages set on the two objects for which you want to retrieve the minimum spacing.

Arguments

d_techID	The database identifier of the technology library.
t_layerName	The layer name.
?constraintGroup *t_constrGroupName*	The constraint group in which minVoltageSpacing is defined. This is a user-defined setup constraint group. The default value is foundry.
?voltage1 *f_voltage1Val*	The voltage value of the first object.
?voltage2 *f_voltage2Val*	The voltage value of the second object.

Value Returned

f_minVolSpaceVal	The minimum voltage-dependent spacing value.
nil	The minimum voltage-dependent spacing is not defined for the specified layer or for the specified arguments.

Examples

```
cv = geGetEditCellView()
tech = techGetTechFile(cv)
drdGetMinVoltageSpacing(tech "Metal6" ?voltage1 0.4 ?voltage2 0.2)
=> 0.1
```

Returns the minimum spacing required in any direction between two Metal6 shapes with a voltage difference of 0.2 (0.4 - 0.2) between them.

```
drdGetMinVoltageSpacing(tech "Metal2" ?voltage1 0.4 ?voltage2 0.2 ?direction
"horizontal")
=> 0.15
```

Returns the minimum spacing required in the horizontal direction between two Metal2 shapes with a voltage difference of 0.2 (0.4 - 0.2) between them.

drdIsPixelThresholdEnabled

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

Description

Indicates whether the pixel threshold is enabled or disabled.

Arguments

None

Value Returned

`t`	The pixel threshold is enabled.
`nil`	The pixel threshold is disabled.

Example

```
drdEnablePixelThreshold(!drdIsPixelThresholdEnabled())
```

Turns on the pixel threshold if it is currently disabled. If found enabled, the pixel threshold is turned off.For more information, see drdEnablePixelThreshold.

drdLayerGroupHeaderClicked

```
drdLayerGroupHeaderClicked(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Changes the state of the check boxes for all the layer categories listed in the *Layer Category* column on the *Filters* tab of the DRD Options form.

Arguments

x_columnIndex	Column index; `0` indicates the *Layer Category* column, `1` indicates the column under which the check boxes for the layer categories are displayed.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdLayerGroupHeaderClicked(0 window(2))
```

It performs no action.

```
drdLayerGroupHeaderClicked(1 window(2))
```

It toggles the state of the check boxes for all the layer categories.

drdLayerHeaderClicked

```
drdLayerHeaderClicked(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Changes the state of the check boxes for all the layers listed in the *Layer* column or sorts the layers alphabetically, on the *Filters* tab of the DRD Options form.

Arguments

x_columnIndex	Column index; `0` indicates none, `1` indicates the *Layer* column, and `2` indicates the *S* column.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdLayerHeaderClicked(0 window(2))
```

It performs no action.

```
drdLayerHeaderClicked(1 window(2))
```

It sorts the layers alphabetically.

```
drdLayerHeaderClicked(2 window(2))
```

It toggles the state of the check boxes in the *S* column.

drdLayerItemClicked

```
drdLayerItemClicked(
     t_layerName
     x_columnIndex
     x_stateValue
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of a layer on the *Filters* tab of the DRD Options form.

Arguments

t_layerName	Name of the layer.
x_columnIndex	Column index; `0` indicates none, `1` indicates the *Layer* column, and `2` indicates the *S* column.
x_stateValue	Value indicating the selection states of a layer; `0` for deselection, `1` for partial selection, and `2` for complete selection.
w_windowId	The current window ID.

Value Returned

`t`	The state of the layer was changed.
`nil`	The state of the layer could not be changed.

Examples

```
drdLayerItemClicked("Via8" 2 0 window(2))
```

It deselects the Via8 layer.

```
drdLayerItemClicked("Via6" 2 2 window(2))
```

It selects the Via6 layer.

drdListConstraintCategoryRules

```
drdListConstraintCategoryRules(
     [ ?cv d_cellViewID ]
     [ ?outputFile s_fileName ]
     [ ?active { t | nil } ]
     [ ?splitTF { t | nil } ]
     [ ?csv { t | nil } ]
     )
     => t / nil
```

Description

Lists constraints supported by DRD in the version of the software that is currently running. The constraints' information can be saved in an ASCII file or listed in CIW, in a tabular format or as a comma-separated list.

The information is organized into the following columns:

- **Category:** Lists the category to which the constraint belongs
- **Rule Name:** Lists the constraint name
- **Technology File Name:** Lists the name of the technology file in which the constraint is defined
- **Notify:** Displays an *N* if the constraint is supported in *Notify* mode
- **Enforce:** Displays an *E* if the constraint is supported in *Enforce* mode
- **PostEdit:** Displays a *P* if the constraint is supported in *Post-Edit* mode
- **Batch:** Displays a *B* if the constraint is supported in *Batch-Check* mode

Arguments

`?cv` *`d_cellViewID`*	Specifies the cellview to use. The default is the current cellview.
`?outputFile` *`s_fileName`*	Specifies the output filename. If the output filename is not specified, the constraints are listed in CIW and the current log file.
`?active`	Determines which constraints are listed. If you specify `?active nil`, all constraints that DRD supports in the version of the software running currently are listed. If you run the function without specifying this parameter or specify `?active t`, the output comprises only those supported constraints that are found in the technology file. The default is `t`.
`?splitTF`	Determines whether the technology filename is displayed separately or displayed in parenthesis together with the constraint name. The default is `t`, which lists the technology filename separately. If set to `nil`, the technology filename is displayed in parenthesis together with the constraint name.

`?csv`	Determines whether the output should be generated as a comma-separated list, with each value enclosed in double quotes, or in a tabular format.
	
	If set to `t`, the output is generated as a comma-separated list as shown below:
	
	`"Spacing","minSpacing","gpdk090","N","E","P","B"`
	
	`"Width","minCornerToCornerDistance","N","","P","B"`
	
	The default is `nil`, which lists the output in a tabular format.

Value Returned

`t`	Constraints were successfully listed.
`nil`	Constraints could not be listed.

Examples

```
drdListConstraintCategoryRules(?cv cvID ?csv t ?outputFile "/home/boyardee/
rules.csv")
```

Saves the constraints defined in the technology file, in CSV format, in a file named `rules.csv`.

```
drdListConstraintCategoryRules(?active nil)
```

Lists all supported constraints in a tabular format in CIW and in the log file for the current session.

```
drdListConstraintCategoryRules(?cv geGetEditCellView() ?outputFile
"./drdConstraints.txt"  ?active nil)
```

Returns all the constraints supported by DRD, whether or not they are defined in the technology library. The output is saved in a file named `drdConstraints.txt`, as shown:

```
Area              minArea                     gpdk045  N   P B
Area              minAreaEdgeLength                    N   P B
Area              minHoleArea                 gpdk045  N   P B
…
…
Complex Spacing   minEndOfLineCutSpacing                   P B
Complex Spacing   minEndOfLineExtensionSpacing             P B
Complex Spacing   endOfLineKeepout                       N   P B
…
…
Density           minDensity                  gpdk045        b
Density           maxDensity                  gpdk045        b
Density           maxDiffDensity                             b
…
…
Spacing           allowedSpacingRanges                 N   P B
Spacing           minCenterToCenterSpacing             N   P B
Spacing           minCutClassSpacing                   N E P B
Spacing           minCutRoutingSpacing                 N E P B
…
…
```

```
Via             largeRectViaArrayAllowed                P B
Via             minLargeViaArrayCutSpacing              P B
Via             minLargeViaArraySpacing                 P B
…
…
Width           minDiagonalWidth          gpdk045  N    P B
Width           protrusionWidth                        P B
Width           minWidth                  gpdk045  N    P B
Width           maxWidth                  gpdk045  N    P B
```

drdOptionsSet

```
drdOptionsSet(
     t_fieldName
     g_state
     )
     => t / nil
```

Description

Sets default states for fields in the configuration file for the DRD Options form.

You can save settings from the DRD Options form to a configuration file and, conversely, load the settings from the configuration file to the form. For more information, see DRD Configuration File in *Virtuoso Design Rule Driven Editing User Guide*.

Arguments

t_fieldName	Name of the field as mentioned in the configuration file.
g_state	Indicates the state of the specified field. It can have one of these values: ■ `t`: enabled ■ `nil`: disabled

Value Returned

`t`	The state was set.
`nil`	The state could not be set.

Examples

```
drdOptionsSet("interactiveNotifyEnabled" t)
```

Enables this option in the DRD Option form: *Interactive – Notify – Enabled*.

```
drdOptionsSet("interactiveNotifyEnabled" nil)
```

Disables this option in the DRD Option form: *Interactive – Notify – Enabled*.

drdOptionUpdateConstraint

```
drdOptionUpdateConstraint(
     l_modeCategories
     l_constraints
     g_state
     )
     => t / nil
```

Description

Sets the default state for specified constraints and modes in the configuration file for the Filters tab of the DRD Options form. If the function is run without a maskLayout cellview open, the settings are discarded.

You can save settings from the DRD Options form to a configuration file or, conversely, load the settings from the configuration file to the form. For more information, see DRD Configuration File in *Virtuoso Design Rule Driven Editing User Guide*.

Note: Virtuoso does not consider this function if specified in `.cdsinit` or `.cdsenv`.

Arguments

l_modeCategories	A comma-separated list of one or more of these checking modes: `Notify`, `PostEdit`, `Batch`, `Enforce`, or `all`.
l_constraints	A comma-separated list of constraint names or constraint categories. The supported keywords for the constraint categories are `All`, `Misc`, `Density`, `Length`, `Width`, `Grid`, `Spacing`, `Area`, `Edge Length`, `Extension`, `Complex Spacing`, `Num Cut`, and `Via`. These keywords are case-sensitive and must be used as is.
g_state	Indicates the state of the specified constraints in the specified modes. It can have one of these values: ■ `t`: enabled ■ `nil`: disabled

Value Returned

`t`	The state was set.
`nil`	The state could not be set.

Examples

```
drdOptionUpdateConstraint("Notify" "minClearance,minWidth" t)
```

Enables the `Notify` mode for the `minClearance` and `minWidth` constraints in the Filters tab of the DRD Options form.

```
drdOptionUpdateConstraint("Notify,Enforce" "minClearance,minWidth" nil)
```

Disables the `Notify` and `Enforce` modes for the `minClearance` and `minWidth` constraints in the Filters tab of the DRD Options form.

```
drdOptionUpdateConstraint("Notify,Batch" "Spacing,Width" nil)
```

Disables the `Notify` and `Batch` modes for the `Spacing` and `Width` constraint categories in the Filters tab of the DRD Options form.

drdOptionUpdateLayer

```
drdOptionUpdateLayer(
     l_layers
     g_state
     )
     => t / nil
```

Description

Sets the default state for specified layers in the configuration file for the Filters tab of the DRD Options form. If the function is run without a maskLayout cellview open, the settings are discarded.

You can save settings from the DRD Options form to a configuration file or, conversely, load the settings from the configuration file to the form. For more information, see DRD Configuration File in *Virtuoso Design Rule Driven Editing User Guide*.

Note: Virtuoso does not consider this function if specified in `.cdsinit` or `.cdsenv`.

Arguments

l_layers	A comma-separated list of layer names or layer categories. The supported keywords for the layer categories are `Routing`, `Device`, `Derived`, and `All`. These keywords are case-sensitive and must be used as is.
g_state	Indicates the state of the specified layers. It can have one of these values: ■ `t`: enabled ■ `nil`: disabled

Value Returned

`t`	The state was set.
`nil`	The state could not be set.

Examples

```
drdOptionUpdateLayer("Metal1,Metal2" t)
```

Enables the `Metal1` and `Metal2` layers in the Filters tab of the DRD Options form.

```
drdOptionUpdateLayer("Metal1,Metal2" nil)
```

Disables the `Metal1` and `Metal2` layers in the Filters tab of the DRD Options form.

```
drdOptionUpdateLayer("Routing,Device" nil)
```

Disables the `Routing` and `Device` layer categories in the Filters tab of the DRD Options form.

drdRemoveTarget

```
drdRemoveTarget(
     w_winID
     l_point
     )
     => t / nil
```

Description

Unsets a DRD target that exists at the specified point in the specified window. If multiple DRD targets overlap at the specified point, the DRD target on the active layer is unset.

You can bind a key to the function and use the bindkey during interactive editing to unset DRD targets. For more information, see Unsetting DRD Targets Using the Bindkey and Bindkey Reassignment in *Virtuoso Design Rule Driven Editing User Guide*.

Arguments

w_winID	ID of the layout window in which you want to unset a DRD target.
l_point	Coordinates of the location in the layout window where the DRD target to be unset is located. The coordinates must be specified in database units (dbu).

Value Returned

`t`	The DRD target at the specified location was unset.
`nil`	The DRD target at the specified location was not unset.

Examples

```
drdRemoveTarget(hiGetCurrentWindow() 2000:3000)
drdRemoveTarget(window(5) 15302:1075)
drdRemoveTarget(window(3) list(9000 3000))
```

drdRuleGroupHeaderClicked

```
drdRuleGroupHeaderClicked(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Toggles *Rule Groups* check boxes. The state of the of check boxes in the right panel, *Rule Types* and *Rule IDs* is also toggled.

Arguments

x_columnIndex	The column index.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdRuleGroupHeaderClicked(1 window(2))
```

Toggles the check box states of all *Rule Groups* in left panel and all associated rule types or rule IDs in right panel.

drdRuleIDHeaderClicked

```
drdCheckedRuleGroupItem(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of check boxes of all rule IDs. The state of the associated *Rule Group* check box in the left panel and the associated rule type is also toggled.

Arguments

x_columnIndex	The column index.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdRuleIDHeaderClicked(1 window(2))
```

Toggles the state of the check boxes of all rule IDs in right panel and the associated rule group and rule type.

drdRuleIDItemClicked

```
drdRuleIDItemClicked(
     t_ruleIDName
     x_selectIndex
     x_checkboxState
     w_windowId
     )
     => t / nil
```

Description

Modifies the check box of the rule ID on the left panel and the corresponding rule group and rule type.

Arguments

t_ruleIDName	Name of the rule ID.
x_selectIndex	Value indicating the selection state; `0` for deselection and `1` for selection.
x_checkboxState	Value indicating the state of the check box.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdRuleIDItemClicked("MOM.A.1_3D3V" 1 2 window(2))
```

Modifies the check box state to enabled for `MOM.A.1_3D3V` rule ID on the right panel. The associated rule group and rule type is modified to enabled or partial enabled state, depending on remaining number of disabled rule IDs associated with them.

```
drdRuleIDItemClicked("MOM.A.1_3D3V" 1 0 window(2))
```

Modifies the check box state to disabled for `MOM.A.1_3D3V` rule ID on the right panel. The associated rule group and rule type is also modified.

drdRuleTypeHeaderClicked

```
drdCheckedRuleGroupItem(
     x_columnIndex
     w_windowId
     )
     => t / nil
```

Description

Toggles the state of check boxes of all rule types. The state of associated *Rule Group* check box in the left panel and the associated rule IDs are also toggled.

Arguments

x_columnIndex	The column index.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdRuleTypeHeaderClicked(1 window(2))
```

Toggles the state of the check box of all rule types in the right panel and the associated rule group and rule IDs.

drdRuleTypeItemClicked

```
drdRuleTypeItemClicked(
     t_ruleTypeName
     x_selectIndex
     x_checkboxState
     w_windowId
     )
     => t / nil
```

Description

Modifies the check box of rule type on the left panel and the corresponding rule group and rule ID.

Arguments

t_ruleTypeName	Name of the rule type.
x_selectIndex	Value indicating the selection state; `0` for deselection and `1` for selection.
x_checkboxState	Value indicating the state of the check box.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdRuleTypeItemClicked("equalArea" 1 2 window(2))
```

Modifies the check box state to enabled for `equalArea` rule type on the right panel and the associated rule group and rule ID is modified to enabled or partial enabled state, depending on remaining number of disabled rule IDs associated with them.

```
drdRuleTypeItemClicked("equalArea" 1 0 window(2))
```

Modifies the check box state to disabled for `equalArea` rule type on the right panel and the associated rule group and rule ID is also modified.

drdSelectConstraintGroupItem

```
drdSelectConstraintGroupItem(
     t_constraintCategoryName
     x_selectIndex
     w_windowId
     )
     => t / nil
```

Description

Selects the constraint category in the *Constraint Category* column and displays the corresponding constraints in the *Constraint* column on the *Filters* tab of the DRD Options form.

Arguments

t_constraintCategoryName	Name of the constraint category.
x_selectIndex	Value indicating the selection states of a constraint category; `0` for deselection and `1` for selection.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdSelectConstraintGroupItem("Spacing" 1 window(2))
```

It selects the Spacing category in the *Constraint Category* column and displays the corresponding constraints in the *Constraint* column.

```
drdSelectConstraintGroupItem("Spacing" 0 window(2))
```

It deselects the Spacing category in the *Constraint Category* column.

drdSelectLayerGroupItem

```
drdSelectLayerGroupItem(
     t_layerCategoryName
     x_selectIndex
     w_windowId
     )
     => t / nil
```

Description

Selects the layer category in the *Layer Category* column and displays the corresponding layers in the *Layer* column on the *Filters* tab of the DRD Options form.

Arguments

t_layerCategoryName	Name of the layer category.
x_selectIndex	Value indicating the selection states of a layer category; `0` for deselection and `1` for selection.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdSelectLayerGroupItem("Derived" 1 window(2))
```

It selects the Derived category in the *Layer Category* column and displays the corresponding layers in the *Layer* column.

```
drdSelectLayerGroupItem("Routing" 0 window(2))
```

It deselects the Routing category in the *Layer Category* column.

drdSelectSignOffRuleGroupItem

```
drdSelectSignOffRuleGroupItem(
     t_ruleGroupName
     x_selectIndex
     w_windowId
     )
     => t / nil
```

Description

Selects the sign off rule group category on the left panel and displays the corresponding rule type or rule ID in the right panel.

Arguments

t_ruleGroupName	Name of the rule group.
x_selectIndex	Value indicating the selection state; `0` for deselection and `1` for selection.
w_windowId	The current window ID.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
drdSelectSignOffRuleGroupItem("Area" 1 window(2))
```

Selects the area rule group on left panel and shows its respective rule types or rule IDs in the right panel.

```
drdSelectSignOffRuleGroupItem("ESD" 0 window(2))
```

De-selects the ESD rule group on the left panel.

drdToggleSmartSnapMode

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

Description

Freezes DRC halos, which helps snap objects to the closest edge of the halo or the object.

For more information, see DRD Smart Snapping in *Virtuoso Design Rule Driven Editing User Guide*.

Arguments

None

Value Returned

`t`	The halo was frozen.
`nil`	The halo could not be frozen.

drdToggleSmartSnapModeForDiscreteSpacing

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

Description

Freezes discrete spacing markers, which helps snap objects to the closest marker.

For more information, see Snapping to Halos and Discrete Spacing Values.

Arguments

None

Value Returned

`t`	The marker was frozen.
`nil`	The marker could not be frozen.

drdVerifySelSet

```
drdVerifySelSet(
     [ d_cellViewID ]
     )
     => ld_dbIDs / 0 / -1
```

Description

Checks the objects selected in the specified design for rule violations and generates a marker for each rule violation found. If a design is not specified, it checks the objects selected in the design open in the current window.

Arguments

d_cellViewID	Database ID of the cellview to be checked for rule violations.

Value Returned

ld_dbIDs	List of marker IDs representing the rule violations found.
0	No markers were created.
-1	An error occurred (for example, non-database objects or objects from different databases were specified).

Examples

```
lm = drdVerifySelSet()
numErrs = if(listp(lm) length(lm) 0)
```

Returns the number of violations found for the objects selected in the current cellview.

```
altCVErrs = drdVerifySelSet(geGetEditCellView(window(2)))
```

Returns the markers IDs representing rule violations found for the objects selected in the cellview open in window 2.

leHiBatchChecker

```
leHiBatchChecker(
     [ w_winID ]
     )
     => t / nil
```

Description

Opens the Batch Checker form.

Batch-Check mode is available only in Layout XL and higher tiers. For more information, see Batch Checking Mode in *Virtuoso Design Rule Driven Editing User Guide*.

Argument

w_winID	ID of the window in which you want to open the Batch Checker form.

Value Returned

t	The Batch Checker form was opened.
nil	The Batch Checker form was not opened.

Examples

```
leHiBatchChecker()
leHiBatchChecker(window(3))
```

leToggleDrdMode

```
leToggleDrdMode(
     )
     => t
```

Description

Cycles through DRD modes: Off, Notify, Enforce, and Enforce and Notify.

Arguments

None

Value Returned

`t`	DRD mode was reset to one of the possible values.

Example

```
leToggleDrdMode()
(LE-102620): Change drdMode from Notify to Enforce...
t
```

15

Multi-Patterning Technology Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso® Multi-Patterning Technology (MPT).

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- mptActivate
- mptCheckFlow
- mptCheckHierarchicalLocks
- mptCleanClusters
- mptCleanColoredDataOnSingleMaskLayer
- mptColorRemastering
- mptDeleteClusters
- mptDefineFlow
- mptDoColorChecks
- mptDoToolbarAction
- mptFixCVUncoloredAndUnlocked
- mptGetColoredPurposes
- mptGetColorShiftingLayers
- mptGetDefaultColoringMethod
- mptGetFlowNames
- mptGetFlowSettings
- mptGetLayerColoringMethod

- mptGetLayerDefaultColor
- mptGetLayerDefaultColorSetting
- mptGetLockDefaultColors
- mptGetOutdatedDesigns
- mptGetTrackPatternPattern
- mptGetUpToDateDesigns
- mptHiStitch
- mptHiUnStitch
- mptIsMaskColorShown
- mptLPPMergeToColor
- mptLockAll
- mptMarkersToColoredBlockages
- mptMarkersToMaskColors
- mptPropagateLocks
- mptPropagateSameMaskGroups
- mptPurgePcell
- mptReColor
- mptReColorFromShapes
- mptReColorSelection
- mptReColorWsp
- mptReconstructStitch
- mptReportColoredShapesOnSingleMaskLayer
- mptReportCurrentSettings
- mptSetDefaultColoringMethod
- mptSetFlow
- mptSetLayerColoringMethod
- mptSetLayerDefaultColor

- mptSetLockDefaultColors
- mptSetTrackPatternPattern
- mptSetupComplianceChecker
- mptShowMaskColor
- mptUnlockAll
- mptUnpropagateLocks
- mptUpdateColor

Note: Keyword arguments are supported in MPT SKILL functions.

mptActivate

```
mptActivate(
     g_activate
     )
     => t / nil
```

Description

Specifies whether to switch the Virtuoso Multi-Patterning Technology coloring engine ON or OFF.

Arguments

g_activate	When set to `t`, the coloring engine is switched ON. When set to `nil`, the coloring engine is switched OFF.

Values Returned

`t`	The coloring engine was switched on (or off).
`nil`	The coloring engine could not be switched on (or off).

Example

```
mptActivate( nil )
```

Turns off the coloring engine.

mptCheckFlow

```
mptCheckFlow(
     t_flowName
     )
     => t / nil
```

Description

Checks the flow settings by loading the check procedure of the specified flow or by comparing the current value of the related environment variables against the environment variables set by the set procedure of the flow.

Arguments

t_flowName	Name of the flow to be defined.

Values Returned

t	The flow settings are checked.
nil	The flow settings could not be checked.

Example

```
mptCheckFlow("FullyColoredAndLocked")
 INFO: Flow 'FullyColoredAndLocked' is properly set.
 t
mptCheckFlow("PartiallyColored")
  WARNING: Env mpt.unclusteredShapeColor has a different value ("random") than the
flow value ("asIs").
  WARNING: Env mpt.reColorReadOnlyCellView has a different value (nil) than the
flow value (t).
  WARNING: Env mpt.globalColorShiftingPolicy has a different value ("none") than
the flow value ("cluster").
  WARNING: Env mpt.enableHCLCreationOnPcells has a different value ("level1_pins")
than the flow value ("all").
  WARNING: Env mpt.lockAllHCLPolicy has a different value ("pcell") than the flow
value ("noHCL").
  WARNING: Env mpt.overrideLockOnConnectedShapes has a different value (t) than the
flow value (nil).
  WARNING: Env mpt.autoPropagateLock has a different value (t) than the flow value
(nil).
```

```
  WARNING: Env mpt.allowLockShiftOverride has a different value (t) than the flow
value (nil).
 WARNING: Found 8 discrepancie(s) for flow 'PartiallyColored'
 nil
```

mptCheckHierarchicalLocks

```
mptCheckHierarchicalLocks(
     d_cellViewID
     )
     => t / nil
```

Description

Performs a check for hierarchical color lock (HCL) conflict in the cellview.

Note: The markers can be generated for a read-only cellview.

Arguments

d_cellViewID	The database ID of the cellview.

Values Returned

`t`	HCL conflict check was performed.
`nil`	HCL conflict check was not performed.

Example

```
cv = dbOpenCellViewByType("IC_ADV_12_1" "TOP2" "layout")
mptCheckHierarchicalLocks(cv)
```

Opens the cellview, `IC_ADV_12_1/TOP2/layout`, and performs a check for the HCL conflict in the cellview.

mptCleanClusters

```
mptCleanClusters(
     d_cellViewID
     )
     => t / nil
```

Description

Cleans out-of-date cluster information in the cellview which can help to reduce the design size and/or improve performance.

Arguments

d_cellViewID	The database ID of the cellview.

Values Returned

`t`	Cluster information was cleaned.
`nil`	The cellview does not exist or another error occurred.

Examples

```
mptCleanClusters( geGetEditCellView() )
```

Cleans cluster information in the current edit cellview.

mptCleanColoredDataOnSingleMaskLayer

```
mptCleanColoredDataOnSingleMaskLayer(
     d_cellViewID | g_libName
     t_layerName
     g_doHierarchy
     )
     => t / nil
```

Description

Cleans layer-based color violations on the entire design hierarchy or library.

Arguments

*d_cellViewID	g_libName*	The database ID of the cellview or the library name.
t_layerName	Name of the layer.
g_doHierarchy	Specifies whether to traverse the sub cells.

Values Returned

t	Layer-based color violations were cleaned.
nil	Layer-based color violations were not cleaned.

Examples

```
mptCleanColoredDataOnSingleMaskLayer(geGetEditCellView()~>lib,"Metal1", nil)
```

Cleans layer-based color violations on the current edit cellview.

mptColorRemastering

```
mptColorRemastering(
     d_cellViewID
     ?mode t_mode
     [ ?mappingFile t_mappingFile ]
     [ ?fromView t_fromView ]
     [ ?toView t_toView ]
     [ ?invalidMappingWord t_invalidMappingWord ]
     )
     => t / nil
```

Description

Remasters color-shifted abstract views to corresponding layout views or color-shifted layout views to abstract views.

Arguments

d_cellViewID	The database ID of the cellview.
?mode *t_mode*	Specifies the color re-mastering mode. Valid values: `abstractToLayout`, `layoutToAbstract`
?mappingFile *t_mappingFile*	The name of the mapping file.
?fromView *t_fromView*	The name of the view to that is to be remastered.
?toView *t_toView*	The name of the view where the remastered view is saved.
?invalidMappingWord *t_invalidMappingWord*	The word to be used to identify invalid mapping.

Values Returned

`t`	Color re-mastering is successful.
`nil`	Color re-mastering is unsuccessful.

Examples

```
mptColorRemastering( geGetEditCellView() ?mode "abstractToLayout")
```

Remasters abstract views to layout views on the current edited cellview.

mptDeleteClusters

```
mptDeleteClusters(
     d_cellViewID
     [ ?layers l_layers ]
     [ ?checkOnly g_checkOnly ]
     [ ?thruHierarchy g_thruHierarchy ]
     [ ?openForEdit g_openForEdit ]
     )
     => t / nil
```

Description

Removes coloring clusters in the specified cellview.

Arguments

d_cellViewID	The database ID of the cellview.
?layers *l_layers*	List of layers from which to remove the coloring clusters. If `nil`, the coloring clusters are removed from all layers. The default is `nil`.
?checkOnly *g_checkOnly*	Specifies whether the cellview has coloring clusters to be deleted. The default is `nil`.
?thruHierarchy *g_thruHierarchy*	Specifies whether the coloring clusters must be deleted from the top-level cellview and cellviews within the hierarchy. The default is `nil`.
?openForEdit *g_openForEdit*	Specifies whether a cellview has clusters to be deleted, but it is read-only. If `t`, opens for edits, read-only cellviews that have clusters to be deleted. The default is `nil`.

Values Returned

`t`	Coloring clusters were removed.
`nil`	Coloring clusters were not removed.

Examples

```
mptDeleteClusters( geGetEditCellView() )
```

Removes top-level coloring clusters on all the layers for the current edit cellview.

mptDefineFlow

```
mptDefineFlow(
     t_flowName
     s_setFunction
     [ s_checkFunction ]
     )
     => t / nil
```

Description

Registers a flow by linking the flow name to a set and a check procedure. If the check procedure is not defined, the check defined in the `cdsenv` file is run. This function checks the environment variables defined in the set function against their current values.

Note: The `s_setFunction` and `s_checkFunction` procedures should be defined in the current virtuoso session before the `mptDefineFlow` calls these procedures.

Arguments

t_flowName	Name of the flow to be defined.
s_setFunction	Name of the procedure used to apply the flow settings.
s_checkFunction	Name of the procedure used to check the flow settings. This is an optional argument.

Values Returned

`t`	The flow is defined.
`nil`	The flow is not defined.

Example

```
load "mptCustomFlow.il"
mptDefineFlow( "custom" 'TacticalFlow_from_CDN_SETUP
'TacticalFlow_from_CDN_CHECK)
```

mptDoColorChecks

```
mptDoColorChecks(
     d_cellViewID
     l_layers
     l_region
     g_doColorShorts
     g_doUncoloredShapes
     g_doUnlockedShapes
     g_doColorability
     )
     => t / nil
```

Description

Performs coloring checks in the specified cellview.

Note: The markers can be generated for a read-only cellview.

Arguments

d_cellViewID	The database ID of the cellview.
l_layers	List of layers to check for coloring. If `nil`, coloring is checked in all layers.
l_region	The region to check for coloring. If `nil`, coloring is checked in the entire design.
g_doColorShorts	Specifies whether the shorts are reported in the coloring check. If `t`, color shorts are reported.
g_doUncoloredShapes	Specifies whether the uncolored shapes are reported in the coloring check. If `t`, uncolored shapes are reported.
g_doUnlockedShapes	Specifies whether the unlocked shapes are reported in the coloring check. If `t`, unlocked shapes are reported.
g_doColorability	Specifies whether the colorability is reported in the coloring check. If `t`, colorability is reported.

Values Returned

`t`	Coloring check was performed.
`nil`	Coloring check was not performed.

Examples

```
mptDoColorChecks(deGetCellView() nil nil t t t t)
```

Performs all the color checks in the design.

```
mptDoColorChecks(deGetCellView() list( "Metal1" "Metal3" ) '((-0.084 1.985) (1.035
3.256)) t t nil nil)
```

Performs the color checks on the `Metal1` and `Metal3` layers in the specified region.

mptDoToolbarAction

```
mptDoToolbarAction(
     g_toolbarFunc
     )
     => t / nil
```

Description

Runs the specified Multiple Patterning toolbar function.

Arguments

g_toolbarFunc Specifies the toolbar function.

`"ColorEngineSwitch"`	Toggles the color engine on and off. Related function: mptActivate
`"ShowHideColor"`	Toggles colors on and off in the canvas. Related function: mptShowMaskColor
`"ShiftColor"`	Shifts the color of the selected shapes, as described in Color Shifting in *Virtuoso Multi-Patterning Technology User Guide*.
`"ReColorAll"`	Opens the Recolor All form. Related function: mptReColor
`"HierarchicalColorLockingCheck"`	Checks for color conflicts related to color locking and creates markers in the canvas that are listed in the Annotation Browser in the *MPT* grouping of the *Misc* tab, as described in Hierarchical Color Locking Check in *Virtuoso Multi-Patterning Technology User Guide*.
`"LockCurrent"`	Locks the selected shapes to their current color, or, if no shapes are preselected, locks shapes to their current color as they are selected.

`"LockColor1"`	Locks the selected shapes to mask1Color, or, if no shapes are preselected, locks shapes to mask1Color as they are selected.
`"LockColor2"`	Locks the selected shapes to mask2Color, or, if no shapes are preselected, locks shapes to mask2Color as they are selected.
`"LockColor3"`	Locks the selected shapes to mask3Color, or, if no shapes are preselected, locks shapes to mask3Color as they are selected.
`"LockColor4"`	Locks the selected shapes to mask4Color, or, if no shapes are preselected, locks shapes to mask4Color as they are selected.
`"Unlock"`	Sets the color state for the selected shapes to unlocked, or, if no shapes are preselected, sets the color state for shapes to unlocked as they are selected.
`"LockAll"`	Opens the Lock All form.
`"UnlockAll"`	Opens the Unlock All form.
`"PropagateLocks"`	Opens the Propagate Locks form. Related function: mptPropagateLocks
`"DeleteColor"`	Removes the locked and unlocked color from shapes in the selected set, or, if no shapes are preselected, removes the locked and unlocked color from shapes as they are selected.
`"DeleteAllColors"`	Opens the Delete All Colors form.Multiple Patterning Options
`"MultiplePatterning"`	Opens the form.

Values Returned

`t`	The Multiple Patterning toolbar action was completed.
`nil`	The toolbar action was invalid and was not run.

Examples

```
mptDoToolbarAction( "UnlockAll" )
```

Opens the Unlock All form.

```
mptDoToolbarAction( "ShiftColor" )
```

Shifts the color of the selected shapes.

mptFixCVUncoloredAndUnlocked

```
mptFixCVUncoloredAndUnlocked(
     d_cellViewID
     [ l_layers
     [ l_region
     [ g_fixUncolored
     [ g_fixUnlocked ]]]]
     )
     => t / nil
```

Description

Searches for uncolored and unlocked, top-level shapes or vias, in the specified cellview and colors and locks them, respectively.

Note: The `mptFixCVUncoloredAndUnlocked` function returns the same results as the functions, mptReColor and mptLockAll at the top level, but is intended for a smaller set of shapes. This function has better performance when there are fewer shapes that need to be colored and locked. It is useful when all metal layers are already colored and locked, but via `cutlayers` still need to be colored and locked.

Arguments

d_cellViewID	The database ID of the cellview.
l_layers	A list of layer names. The default value is `nil`, which implies that all colorable layers are searched.
l_region	The region or design area to be searched. The default value is `nil`, which implies that the entire cellview is searched.
g_fixUncolored	Specifies if uncolored shapes and vias are colored. If `t`, the uncolored shapes and vias are colored. If `nil`, the uncolored shapes and vias are not colored. The default value is `t`.
g_fixUnlocked	Specifies that unlocked shapes and vias are locked, when set to `t`, the default value. If set to `nil`, unlocked shapes and vias are not locked.

Values Returned

`t`	The uncolored or unlocked shapes or vias are colored or locked.
`nil`	The function was not successful.

Examples

```
mptFixCVUncoloredAndUnlocked(geGetEditCellView() list("M1") nil t nil)
```

Searches the uncolored objects, shapes or vias, on layer `M1` for the currently edited cell view.

mptGetColoredPurposes

```
mptGetColoredPurposes(
     d_cellViewID
     )
     => l_lpp / nil
```

Description

Returns a list of the colored layer purpose pairs for the cellview.

Arguments

d_cellViewID	The database ID of the cellview.

Values Returned

l_lpp	List of the colored layer purpose pairs.
nil	The cellview was not found.

Examples

```
mptGetColoredPurposes( geGetEditCellView() )
```

Returns the list of colored layer purpose pairs for the current edit cellview.

mptGetColorShiftingLayers

```
mptGetColorShiftingLayers(
     d_cellViewID
     )
     => l_layers / nil
```

Description

Returns a list of valid shifting layers set by the environment variable .

Arguments

d_cellViewID	Database ID of the cellview.

Values Returned

l_layers	List of valid shifting layers set by the environment variable colorShiftingLayers.
nil	The list of valid shifting layers is not returned.

Examples

```
mptGetColorShiftingLayers(geGetEditCellView())
```

Returns the list of valid shifting layers set by the environment variable, colorShiftingLayers.

mptGetDefaultColoringMethod

```
mptGetDefaultColoringMethod(
     )
     => t_coloringMethod
```

Description

Returns the session default coloring method that was set using mptSetDefaultColoringMethod.

Argument

None

Values Returned

t_coloringMethod	The session default coloring method. Valid values: `"connectedShapes"`, `"colorSpacing"`

Example

```
mptSetDefaultColoringMethod( "connectedShapes" )
=> t
mptGetDefaultColoringMethod()
=> "connectedShapes"
```

Returns the session default coloring method that was set using mptSetDefaultColoringMethod. The returned values are shown in blue.

mptGetFlowNames

```
mptGetFlowNames(
     )
     => l_flowName
```

Description

Lists all available flow names that have been defined. It includes pre-defined and custom flow names.

Arguments

None

Values Returned

l_flowName	List of flow names.

Example

```
mptGetFlowNames()
  Available MPT flows:
   FullyColoredAndLocked, PartiallyColored (pre-defined)
   custom (user-defined)
 ("FullyColoredAndLocked" "PartiallyColored" "custom")
```

mptGetFlowSettings

```
mptGetFlowSettings(
     t_flowName
     )
     => t
```

Description

Reports, the procedure used to set a flow, in the CIW and output.

Arguments

t_flowName	Name of the flow to be defined.

Values Returned

t	The flow is reported and a short report is displayed in the CIW.

Example

```
mptGetFlowSettings("FullyColoredAndLocked")
INFO: Displaying 'FullyColoredAndLocked' flow settings
 envSetVal("mpt" "coloringEngineEnabled" 'boolean t)
 envSetVal("mpt" "unclusteredShapeColor" 'cyclic "random")
 envSetVal("mpt" "defaultColoringMethod" 'cyclic "colorSpacing")
 envSetVal("mpt" "reColorReadOnlyCellView" 'boolean nil)

 envSetVal("mpt" "dontColorPCells" 'boolean nil)
 envSetVal("mpt" "forcePcellRecolorOnEval" 'boolean t)

 envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "none")
 envSetVal("mpt" "globalColorShiftingPolicyForPcells" 'cyclic "cluster")

 envSetVal("mpt" "enableHCLCreation" 'cyclic "all")
 envSetVal("mpt" "enableHCLCreationOnPcells" 'cyclic "level1_pins")
 envSetVal("mpt" "lockAllHCLPolicy" 'cyclic "pcell")

 envSetVal("mpt" "propagateLocksToConnectedShapes" 'boolean t)
 envSetVal("mpt" "overrideLockOnConnectedShapes" 'boolean t)
```

```
 envSetVal("mpt" "autoPropagateLock" 'boolean t)
 envSetVal("mpt" "allowLockShiftOverride"'boolean t)
 envSetVal("mpt" "propagateAnySameMaskState" 'boolean t)

 envSetVal("cdba" "copyMPAttributes" 'boolean t)
t
```

mptGetLayerColoringMethod

```
mptGetLayerColoringMethod(
     d_cellViewID
     [ t_layerName ]
     )
     => t_coloringMethod / t_SDLcolorMethod / nil
```

Description

Returns either the coloring method for the given layer, or the coloring method for the session, the design, and layers with a specified coloring method.

Arguments

d_cellViewID	The database ID of the cellview.
t_layerName	Name of the layer. If not specified, the coloring method for the session, the design, and layers with a specified coloring method are reported.

Values Returned

t_coloringMethod	The coloring method for the given layer. Valid values: `"connectedShapes"`, `"colorSpacing"`
t_SDLcolorMethod	The coloring method for the session, the design, and layers with a specified coloring method.
`nil`	The coloring method was not returned due to an invalid cellview, layer name, or some other error.

Examples

```
mptGetLayerColoringMethod( cv )
Session coloring method : connectedShapes
Design coloring method : connectedShapes
Layer Metal1 : colorSpacing
```

Returns the coloring method for the session, the design given by `cv`, and layers for which the coloring method is set. In this example, the `colorSpacing` coloring method was set for `Metal1` using mptSetLayerColoringMethod, and both the session and design coloring methods are `connectedShapes`. The returned values are shown in blue.

```
mptGetLayerColoringMethod(cv "Metal1")
"colorSpacing"
```

Returns the coloring method for layer `Metal1` of the design given by `cv`.

mptGetLayerDefaultColor

```
mptGetLayerDefaultColor(
     d_techFileID
     [ t_layerName ]
     )
     => t_color / l_layerColor / nil
```

Description

Returns either the default color for the given layer, or a list of layer-default color pairs.

Arguments

d_techFileID	Technology database ID.
t_layerName	Name of the layer. If not specified, a list of layer-default color pairs is output for the layers with default colors that were set using mptSetLayerDefaultColor.

Values Returned

t_color	The default color for the given layer.
l_layerColor	List of layer names and default color pairs for layers with default colors that were set using mptSetLayerDefaultColor.
nil	The default color was not returned due to an invalid technology database ID, layer name, or some other error.

Example

```
tfID=techGetTechFile( geGetEditCellView()~>lib )
mptGetLayerDefaultColor( tfID "m1" )
"mask1Color"
```

Returns the default color for layer `m1` that was set using mptSetLayerDefaultColor. The returned value is shown in blue.

```
mptGetLayerDefaultColor( tfID )
(("m1" "mask1Color")("m2" "mask2Color"))
```

Returns a list of the layer and default color pairs for the layers for which the default color was set using mptSetLayerDefaultColor.

mptGetLayerDefaultColorSetting

```
mptGetLayerDefaultColorSetting(
     d_techFileID
     [ t_layerName ]
     [ b_getAll ]
     )
     => t_color / l_layerColor / l_ColorSettings / nil
```

Description

Returns the default color setting for the specified layer.

Arguments

d_techFileID	Technology database ID.
t_layerName	Name of the layer.
b_getAll	Gets all valid setting of the specified layer.

Values Returned

t_color	The color for the specified layer.
l_layerColor	List of layer names and color pairs for if layer name is not specified.
l_ColorSettings	List of all possible color settings for the layer if a layer name is specified and the value for the *b_getAll* argument is `t`.
`nil`	The default color was not returned due to an invalid technology database ID, layer name, or some other error.

Example

```
techFileId = techGetTechFile(geGetEditCellView()~>lib)
mptSetLayerDefaultColor(techFileId "m1" "mask1Color")
mptSetLayerDefaultColor(techFileId "m2" "mask2Color")

mptGetLayerDefaultColorSetting(techFileId "m1")
"mask1Color"

mptGetLayerDefaultColorSetting(techFileId)
```

```
(("m1" "mask1Color")("m2" "mask2Color"))


mptGetLayerDefaultColorSetting(techFileId "m1" t)
("No Setting" "(Always) Mask1Color" "(Always) Mask2Color" "(Always) Mask3Color"
"Alternating")
```

mptGetLockDefaultColors

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

Description

Indicates whether the layer default color specified by mptSetLayerDefaultColor will be locked when assigned to a shape.

Arguments

None

Values Returned

`t`	The layer default color will be locked when assigned to a shape.
`nil`	The layer default color will not be locked when assigned to a shape.

Example

```
mptGetLockDefaultColors()
t
```

Indicates whether the layer default color will be locked when assigned to a shape. In this example, the layer default color will be locked.

mptGetOutdatedDesigns

```
mptGetOutdatedDesigns(
     d_cellViewID
     [ t_mode ]
     )
     => l_outdated / nil
```

Description

Returns a list of the cellviews in the design hierarchy for which the color information is outdated. The list can optionally specify the outdated layers for each cellview in the returned list.

Outdated cellviews have one or more of the following characteristics:

- The cellview was edited while the color engine was off.
- Color information was updated more recently for data at a lower level of the hierarchy.
- Data at a lower level of the hierarchy was edited while the color engine was off.

Arguments

d_cellViewID	The database ID of the top-level cellview to be checked.

t_mode	Specifies the contents for the list to be returned.
	Valid values:
	■ `"perDesign"`: A list of the cellviews in the hierarchy (represented by database IDs) in bottom-up order that have outdated color information. This is the default. For example:
	`list(`*d_cellViewID2 d_cellViewID1*`)`
	■ `"perLayer"`: A list of cellviews in the hierarchy (represented by database IDs) in bottom-up order, and layer list pairs. The layers in the list for the associated cellview have outdated color information. For example:
	`list((`*d_cellViewID2* `list(`*g_layerNum …*`))`
	`(`*d_cellViewID1 list(g_layerNum …*`)))`

Values Returned

l_outDated	The list of cellview-layer list pairs or cellviews in the hierarchy with outdated color information.
`nil`	None of the cellviews for this design has outdated color information.

Examples

```
mptGetOutdatedDesigns( geGetEditCellView() "perDesign" )
(db:0x2d54ca9a db:2d54971a)
```

Returns the list of cellviews in the hierarchy, in bottom-up order that have outdated color information.

```
mptGetOutdatedDesigns( geGetEditCellView() "perLayer" )
((db:0x2d54ca9a
    (34)
    )
    (db:2d54971a
    (30 34)
    )
)
```

Returns the list of cellview-layer list pairs, given in bottom-up order for the current edit cellview. The layers listed for each cellview have outdated color information. In this example, the cellview with the ID of `db:0x2d54ca9a` has one layer number 34 with outdated color information, and the cellview with the ID of `db:2d54971a` has two layers, layer numbers `30` and 34, with outdated color information. In this case, the second cellview in the list is the top-level cellview.

Related Topic

mptGetUpToDateDesigns

mptGetTrackPatternPattern

```
mptGetTrackPatternPattern(
     d_trackPatternID
     )
     => l_colorPattern / nil
```

Description

Returns a list of the color patterns for the specified track pattern.

Argument

d_trackPatternID

The database ID of the track pattern.

Values Returned

l_colorPattern	The list of color patterns for the specified track pattern.
nil	No color patterns returned for the specified track pattern.

Example

```
cv =geGetEditCellView()
mptGetTrackPatternPattern( tp0 )
    => "1234"
```

Returns the list of color patterns for the track pattern, `tp0`.

mptGetUpToDateDesigns

```
mptGetUpToDateDesigns(
     d_cellViewID
     )
     => l_upToDate / nil
```

Description

Returns a list of the cellviews in the hierarchy for which the color information is up to date.

Argument

d_cellViewID	The database ID of the top-level cellview to be checked.

Values Returned

l_upToDate	The list of cellviews, represented as database IDs, with color information that is up to date.
nil	None of the cellviews for this design has up-to-date color information.

Example

```
mptGetUpToDateDesigns( geGetEditCellView() )
(db:0x2d54ca9a db:2d54971a)
```

Returns the list of cellviews in the hierarchy that have up-to-date color information.

Related Topic

mptGetOutdatedDesigns

mptHiStitch

```
mptHiStitch(
     )
     => nil
```

Description

Enables stitch mode to fix odd loop color violations by breaking shapes into two overlapping (stitched) shapes. The first click in the layout selects the shape to be stitched, and the second click sets the stitch location.

Arguments

None

Value Returned

`nil`	Stitch mode is no longer enabled.

Example

```
mptHiStitch()
```

Enables stitch mode.

mptHiUnStitch

```
mptHiUnStitch(
     )
     => nil
```

Description

Enables unstitch mode to revert stitches that were created while in stitch mode (enabled by mptHiStitch). Click a stitched shape in the layout to remove the stitch.

Arguments

None

Value Returned

`nil`	Unstitch mode is no longer enabled.

Example

```
mptHiUnStitch()
```

Enables unstitch mode.

mptIsMaskColorShown

```
mptIsMaskColorShown(
     t_layerName
     x_maskNumber
     )
     => t / nil
```

Description

Checks whether the specified mask color is visible for the layer.

Arguments

t_layerName	Name of the layer.
x_maskNumber	Mask color is 1, 2, and so on, up to the number of masks defined in the technology library for the layer.

Value Returned

`t`	The specified mask color is visible for the layer.
`nil`	The specified mask color is not visible (hidden) for the layer.

Example

```
mptIsMaskColorShown( "Metal1" 2 )
```

Checks whether `mask2Color` is visible for the `Metal1` layer.

mptLPPMergeToColor

```
mptLPPMergeToColor(
     d_cellviewID
     g_libName
     g_viewName
     l_mapping
     [ g_colorState ]
     )
     => t / nil
```

Description

Converts the layout into Virtuoso multi-pattern conformance format for designs that use two or more separate layer-purpose pairs to represent multi-patterning mask information, so that Virtuoso coloring engine can perform further operations on it. You can choose to write the conversion result into a different library or/and a different view. In this case, new cells with the same cell name and the library or/and view name that you specify are used. The original cell is not changed. If you choose to write the result to the same cell, the original layer-purpose pair data is removed. This function is performed on the entire hierarchy starting from the specified cell.

Important

Net information and any previous mask coloring are not copied when output is to an existing cellview that is different from the source cellview.

Arguments

d_cellviewID	The database ID of the top-level cellview where the data conversion is to be started. When this function is applied, it covers the entire hierarchy tree and converts each cell as necessary.
g_libName	Library name. If this argument value is a string, the result is written to the library with this name. If the argument value is `nil`, the result is written to the same library. If the *g_viewName* argument is `nil` as well, the result is written to the same cell.
g_viewName	View name. If the argument value is a string, the result is written to the view with this name. If the argument value is `nil`, the result is written to the same view. If the *g_libName* argument is `nil` as well, the result is written to the same cell.
l_mapping	A list specifying the mapping of layer-purpose pair to color information. Each element in this list contains a list of four or more string values (depending on the number of masks), which are in this order: *layer_name*, *result_purpose_name*,and a *color_*<mask number>*_purpose_name* for each mask on the layer, in numeric order. A two-mask layer would have a list of four values, and a three-mask layer would have a list of five values.
g_colorState	Specifies the color state of the output layer-purpose pair. If `t`, the output layer-purpose pair will be color locked. If `nil` (the default), the output layer-purpose pair will not be color locked.

Values Returned

`t`	The data is converted.
`nil`	The data is not converted.

Examples

```
mptLPPMergeToColor( geGetEditCellView() nil nil list( list( "Metal1" "drawing"
"dr1" "dr2" "dr3" ) )
```

The following tasks are performed for the three-mask `Metal1` layer:

- Hierarchically converts the layer-purpose-pair based multi-patterning design into Virtuoso standard multi-patterning data, starting from the cellview currently being edited.
- Writes the result to the same cell.
- Moves all shapes on `(Metal1, dr1)` to `(Metal1, drawing)` with color set to `1`.
- Moves all shapes on `(Metal1, dr2)` to `(Metal1, drawing)` with color set to `2`.
- Moves all shapes on `(Metal1, dr3)` to `(Metal1, drawing)` with color set to `3`.
- Shapes originally on `(Metal1, drawing)` will remain on `(Metal1, drawing)` with color set to `0`.

```
mptLPPMergeToColor( geGetEditCellView() "colorLib" nil list( list( "Metal1"
"drawing" "dr1" "dr2" ) )
```

The following tasks are performed for the two-mask `Metal1` layer:

- Hierarchically converts the layer-purpose-pair based multi-patterning design into Virtuoso standard multi-patterning data, starting from the cellview currently being edited.
- Writes the results to library `colorLib` with the same cell and view name.
- Copies all shapes on `(Metal1, dr1)` from the original cell to `(Metal1, drawing)` with color set to `1` in the destination cell.
- Copies all shapes on `(Metal1, dr2)` from the original cell to `(Metal1, drawing)` with color set to `2` in the destination cell.
- Copies the shapes on `(Metal1, drawing)` from the original cell on `(Metal1, drawing)` with color set to `0` in the destination cell.

mptLockAll

```
mptLockAll(
     d_cellViewID
     [ l_layerNames ]
     )
     => t / nil
```

Description

Locks all the top-level shapes of the specified cellview with their current color. You can optionally lock only the shapes on specific colorable layers. Gray color shapes are not locked.

Arguments

d_cellViewID	The database ID of the cellview to be processed.
l_layerNames	Processes only the shapes on the colorable layers in this list.

Value Returned

`t`	The shapes are locked to their current color.
`nil`	The shapes are not locked due to an error (for example, invalid cellview ID or other invalid argument).

Example

```
mptLockAll( geGetEditCellView() )
```

Locks all the shapes at the current editing hierarchy level to their current colors.

```
mptLockAll( cv list( "Metal1" "Metal2" ) )
```

Locks all the `Metal1` and `Metal2` top-level shapes of the `cv` cellview to their current colors.

Related Topics

Lock and Unlock All

mptUnlockAll

mptMarkersToColoredBlockages

```
mptMarkersToColoredBlockages(
     l_purposes
     d_cellViewID
     )
     => t / nil
```

Description

Transforms the shapes on the specified purposes of the colored layers in the given cellview to colored blockages on the same layer. This function works hierarchically on a layout.

Arguments

l_purposes	List of purposes representing mask1Color, mask2Color, mask3Color, and mask4Color markers, in that order. The mask4Color purpose is required only for designs with quadruple patterning layers. The mask3Color purpose is required only for designs with triple patterning and/or quadruple patterning layers.
d_cellViewID	The database ID of the cellview on which the marker to colored blockage transformation is to be performed hierarchically.

Values Returned

t	The shapes in the cellview on the purposes specified in the list are transformed to colored blockages on the same layer.
nil	No transformation occurs.

Examples

```
mptMarkersToColoredBlockages( list("blockage1" "blockage2") geGetEditCellView() )
```

Transforms the shapes on `blockage1` and `blockage2` purposes to blockage shapes on mask1Color and mask2Color, respectively, for the current cellview that has double patterning layers.

```
mptMarkersToColoredBlockages( list( "blockage1" "blockage2" "blockage3" )
geGetEditCellView() )
```

Transforms the shapes on `blockage1`, `blockage2`, and `blockage3` purposes to blockage shapes on mask1Color, mask2Color, and mask3Color, respectively, for the current cellview that has triple patterning layers.

```
mptMarkersToColoredBlockages( list( "blockage1" "blockage2" "blockage3"
"blockage4" ) geGetEditCellView() )
```

Transforms the shapes on `blockage1`, `blockage2`, `blockage3`, and `blockage4` purposes to blockage shapes on mask1Color, mask2Color, mask3Color, and mask4Color, respectively, for the current cellview that has quadruple patterning layers.

mptMarkersToMaskColors

```
mptMarkersToMaskColors(
     l_purposes
     g_drawingPurpose
     d_cellViewID
     [ g_locked ]
     [ g_removeMarker ]
     [ g_recolorAll ]
     [ g_save ]
     )
     => t / nil
```

Description

Transforms the shapes in the given cellview to the `drawing` purpose and assigns color to each shape based on its original marker purpose. This function works hierarchically on a layout. By default, the assigned color is locked and the marker purpose shapes are removed.

Arguments

l_purposes	The list of one or more purposes for mask1Color, mask2Color, mask3Color, and mask4Color markers, in that order.
g_drawingPurpose	The purpose of the drawing layer.
d_cellViewID	The database ID of the cellview on which the marker to mask color translation is to be performed hierarchically.
g_locked	Specifies whether the color state is locked on the shapes that are transformed by this function. Valid values: `t` or `nil`. The default is `t`.
g_removeMarker	Specifies whether the marker purpose shapes are removed by this function. Valid values: `t` or `nil`. The default is `t`.
g_recolorAll	Specifies whether all the cellviews are recolored by this function. Valid values: `t` or `nil`. The default is `t`.
g_save	Specifies whether all the modified cellviews in the hierarchy are saved automatically. Valid values: `t` or `nil`. The default is `nil`.

Values Returned

`t`	The color information for the shapes is transformed.
`nil`	The color information for the shapes is not transformed.

Example

```
mptMarkersToMaskColors( list( "boundary" "fill" "hole" ) "drawing"
geGetEditCellView() )
```

Transforms the shapes in the current edit cellview on `boundary`, `fill`, and `hole` purposes to the `drawing` purpose and assigns them mask1Color, mask2Color, and mask3Color, respectively. The color state for the transformed shapes is locked and the marker purpose shapes are removed.

```
mptMarkersToMaskColors( list( "boundary" "hole" ) "drawing" geGetEditCellView()
nil nil)
```

Transforms the shapes in the current edit cellview on `boundary` and `hole` purposes to the `drawing` purpose and assigns them mask1Color and mask2Color, respectively. The color state for the transformed shapes is unlocked and the marker purpose shapes are not removed.

mptPropagateLocks

```
mptPropagateLocks(
     [ d_cellviewID ]
     [ g_convertGroups ]
     )
     => t / nil
```

Description

Propagates all the locks visible from the top level to connected shapes through the hierarchy.

Arguments

d_cellviewID	The database ID of the top-level cellview. If this argument is not given, the current edit cellview is used.
g_convertGroups	When set to `t`, all shapes that are in a color group but are not locked will be locked. This is the default. When set to `nil`, lock propagation on color groups is not performed.

Values Returned

`t`	Lock propagation was successful.
`nil`	Lock propagation was not successful.

Example

```
mptPropagateLocks()
```

Propagates locks for the current edit cellview.

mptPropagateSameMaskGroups

```
mptPropagateSameMaskGroups(
     d_cellViewID
     )
     => t / nil
```

Description

Propagates same mask groups in the given cellview through the hierarchy. This should be run following top-level editing of designs with same mask groups. This ensures that shapes in the hierarchy that are connected to same mask group shapes are included in that mask group.

Arguments

d_cellViewID	The database ID of the cellview.

Values Returned

`t`	Same mask groups are propagated through the hierarchy.
`nil`	Same mask groups are not propagated through the hierarchy.

Example

```
mptPropagateSameMaskGroups( cv )
```

Propagates same mask groups in `cv` through the hierarchy.

mptPurgePcell

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

Description

Removes the color of Pcell or express Pcell colored by the coloring engine. Pcells are purged and the cache of the purged express Pcell is deleted. This is performed hierarchically. Before the color is removed, the coloring engine is deactivated to prevent recoloring of Pcells.

Arguments

None

Values Returned

`t`	Pcell purge was successful.
`nil`	Pcell purge was not successful.

Example

Open a design hierarchy and specify the following command in the CIW:

```
mptPurgePcell()
```

mptReColor

```
mptReColor(
     d_cellViewID
     [ l_layers ]
     [ g_depth ]
     )
     => t / nil
```

Description

Rebuilds the color information for the cellview from the bottom hierarchically, according to the specified depth and layers.

Arguments

d_cellViewID	The database ID of the cellview.
l_layers	List of colorable layer names. If `nil`, all colorable layers are processed.
g_depth	Depth (hierarchy level) for bottom-up processing. If not specified, the entire hierarchy will be processed.

Values Returned

`t`	Recoloring was performed.
`nil`	Recoloring was not successful.

Examples

```
mptReColor( geGetEditCellView() )
```

Recolors the current edit cellview through the entire hierarchy on all colorable layers.

```
mptReColor( geGetEditCellView() list( "m1" "m2" ) )
```

Recolors the current edit cellview through the entire hierarchy on layers `m1` and `m2`.

```
mptReColor( geGetEditCellView() nil 2 )
```

Recolors the current edit cellview bottom-up from hierarchy level 2 on all colorable layers.

mptReColorFromShapes

```
mptReColorFromShapes(
     l_shapes
     )
     => t / nil
```

Description

Recolors the shapes in the list and the shapes that are connected to them. If the coloring method is `colorSpacing`, the color of the shapes that are within the same-mask spacing of the shapes in the list will be updated, according to the clustering algorithm described in Automatic Color Shifting in *Virtuoso Multiple Patterning Technology User Guide*.

Arguments

l_shapes	List of shapes.

Values Returned

`t`	Recoloring was performed.
`nil`	Recoloring was not successful.

Example

```
selectedShapes = setof(obj geGetSelSet() obj~>isShape)
mptReColorFromShapes( selectedShapes )
```

Recolors the shapes in the selection set and the shapes that are connected to them. If the coloring method is `colorSpacing`, the color of the shapes that are within the same-mask spacing of the shapes in the list will also be updated.

mptReColorSelection

```
mptReColorSelection(
     d_cellViewID
     l_selection
     )
     => t / nil
```

Description

Recolors the shapes specified in the selection set in the specified cellview.

Arguments

d_cellViewID	The database ID of the cellview.
l_selection	List of shapes to be recolored

Values Returned

t	Recoloring was performed.
nil	Recoloring was not successful.

Examples

```
mptReColorSelection(cv
geGetObjectSelectedSet(cv)) )
```

Recolors the selection set in the cellview, cv.

mptReColorWsp

```
mptReColorWsp(
     d_cellViewID
     [ ?layers l_layers ]
     [ ?region l_bBox ]
     [ ?grayShapeOnly g_grayShapeOnly ]
     [ ?trackPatterns l_trackPatterns ]
     [ ?shapes l_shapes ]
     [ ?checkWidth g_checkWidth ]
     [ ?vias l_vias ]
     )
     => l_coloredShapes / nil
```

Description

Recolors shapes based on the color of WSPs only.

Arguments

d_cellViewID	The database ID of the cellview.
?layers *l_layers*	List of colorable layer names. If nil, all colorable layers are processed.
?region *l_bBox*	Processes only the shapes overlapping the specified bounding box.
?grayShapeOnly *g_grayShapeOnly*	If t, then only gray shapes are processed. If nil, all shapes are processed
?trackPatterns *l_trackPatterns*	List of track patterns. If nil, all track patterns are processed.
?shapes *l_shapes*	List of shapes to be processed. If nil, all shapes are processed. You can specify *l_layers* or *l_shapes*. If both are specified, no processing happens.
?checkWidth *g_checkWidth*	If t, both the centerline and edges are checked. If nil, only centerline is checked.
?vias *l_vias*	List of vias to be colored.

Values Returned

l_coloredShapes	List of objects colored.
nil	No objects are colored.

Examples

```
mptReColorWsp(geGetEditCellView() ?shapes geGetSelSet())
(db:0x1901bf2d db:0x1901bf2b db:0x1901bf37 db:0x1901bf36 db:0x1901bf28)
```

Returns the list of recolored objects in the selection.

```
mptReColorWsp(geGetEditCellView() ?layers list("M1"))
(db:db:0x1901bf1a db:0x1901bf1d)
```

Returns the list of recolored objects in layer 'M1'.

```
mptReColorWsp(geGetEditCellView() ?layers list("M1") ?checkWidth t)
(db:db:0x1901bf1a)
```

Returns the list of recolored objects in layer 'M1' and both centerline and edge check passed.

```
mptReColorWsp(geGetEditCellView() ?layers list("M1") ?grayShapeOnly t ?checkWidth
t)
```

No objects are recolored based on the criteria.

mptReconstructStitch

```
mptReconstructStitch(
     d_cellViewID
     [ t_purposeName ]
     )
     => t / nil
```

Description

Reconstructs stitches by creating stitch pairs from overlapping shapes on the specified purpose.

Arguments

d_cellViewID	The database ID of the cellview.
t_purposeName	Purpose for the stitch shapes. Default value: `"stitch"`

Values Returned

`t`	Stitches are reconstructed in the cellview.
`nil`	Stitches are not reconstructed in the cellview.

Example

```
mptReconstructStitch( deGetCellView() "stitch" )
```

Reconstructs stitches in the current cellview from shapes on the `stitch` purpose.

mptReportColoredShapesOnSingleMaskLayer

```
mptReportColoredShapesOnSingleMaskLayer(
     d_cellViewID
     t_layerName
     g_doHierarchy
     g_createMarkers
     )
     => t / nil
```

Description

Reports MPT flow settings. Reports the number of shapes, vias, and HCLs that have colored data on the specified layer, with a single mask. This function traverses the design and its sub cells and creates the report. If the `g_createMarkers` argument is set to `t`, markers are created for each error.

Arguments

d_cellViewID	The database ID of the cellview.
t_layerName	Name of the layer.
g_doHierarchy	Specifies whether to traverse the sub cells.
g_createMarkers	Specifies whether the markers are created.

Values Returned

`t`	The report is created.
`nil`	The report is not created.

Examples

```
mptReportColoredShapesOnSingleMaskLayer(cv "M1" t t)
```

Analyzes the top and all the instantiated cells in hierarchy. The number of colored shapes are reported and markers are created with the text `shape has colored data and is on a single mask layer (instance= I2/I1 cell=N10XL/top/layout).`

mptReportCurrentSettings

```
mptReportCurrentSettings(
     d_cellViewID
     [ t_fileName ]
     g_reportDesign
     )
     => t / nil
```

Description

Reports MPT flow settings.

Arguments

d_cellViewID	The database ID of the cellview.
t_fileName	The name is the file to be used to store the report. This is a temporary file.
g_reportDesign	Specifies whether the report design contains MPT data. The default value is `nil`.

Values Returned

`t`	The report is created.
`nil`	The report is not created.

Examples

```
mptReportCurrentSettings(geGetEditCellView())
-----------------------------------------------------
MPT flow report for US_8ths/Asize/symbol
    @(#)$CDS: virtuoso version ICADVM20.1-64b 08/30/2018 09:24 (hc1lnx16) $
    Sep  3 10:02:57 2018
-----------------------------------------------------

Technology file:
    Colorable layer(s):

    MPT constraintGroup: default
        Available MPT constraintGroup(s) in the tech file: none
```

```
Colorable LPPs:

Default Colors:
    none

Coloring Methods:
    Session coloring method : colorSpacing
    Design coloring method : colorSpacing

Main MPT cdsenv:
    Dynamic Coloring/Coloring Engine:t(mpt.coloringEngineEnabled)
    Default shape assignment:"random"(mpt.unclusteredShapeColor)
    ReColor read-only cellViews:nil(mpt.reColorReadOnlyCellView)
    Copy color(makeCell & flatten):t(cdba.copyMPAttributes)
    Lock propagation:   t       (mpt.propagateLocksToConnectedShapes)
    HCL creation during LockAll:"pcell"(mpt.lockAllHCLPolicy)

    Policy for CELL:
        Instance Color Shifting:"none"(mpt.globalColorShiftingPolicy)
        Hierarchical Color Locking:"all"(mpt.enableHCLCreation)

    Policy for PCELL:
        Coloring:               t       (Opposite of mpt.dontColorPCells)
        Recoloring On Evaluation:t(mpt.forcePcellRecolorOnEval)
        Instance Color Shifting:"cluster"(mpt.globalColorShiftingPolicyForPcells)
        Hierarchical Color Locking:"level1_pins"(mpt.enableHCLCreationOnPcells)

Cdsenv with current value different than default:
    mpt allowLockShiftOverridebooleant(default: nil)
    mpt propagateAnySameMaskStatebooleant(default: nil)
    mpt autoPropagateLockbooleant(default: nil)
    mpt forcePcellRecolorOnEvalbooleant(default: nil)
    mpt overrideLockOnConnectedShapesbooleant(default: nil)
    mpt lockAllHCLPolicycyclic"pcell"(default: "noHCL")
    mpt propagateLocksToConnectedShapesbooleant(default: nil)
    mpt coloringEngineEnabledbooleant(default: nil)
    mpt unclusteredShapeColorcyclic"random"(default: "asIs")
    mpt defaultColoringMethodcyclic"colorSpacing"(default: "connectedShapes")
    mpt enableHCLCreationOnPcellscyclic"level1_pins"(default: "all")
```

```
INFO: Report saved to 'mptFlow.rpt'
t
```

mptSetDefaultColoringMethod

```
mptSetDefaultColoringMethod(
     t_coloringMethod
     )
     => t / nil
```

Description

Sets the session default coloring method.

Arguments

t_coloringMethod	Sets the session default coloring method. Valid values: "connectedShapes", "colorSpacing" Default value: "connectedShapes"

Values Returned

t	The default coloring method was set for the current session.
nil	The default coloring method was not set.

Example

```
mptSetDefaultColoringMethod( "colorSpacing" )
```

Sets the session default coloring method to colorSpacing.

mptSetFlow

```
mptSetFlow(
     t_flowName
     )
     => t
```

Description

Applies the flow settings by loading the procedure linked to the flow name.

Arguments

t_flowName	Name of the flow to be defined.

Values Returned

t	The command is successful.
nil	The command is not successful.

Example

```
mptSetFlow("PartiallyColored")
INFO: Applying 'PartiallyColored' flow settings
t
```

mptSetLayerColoringMethod

```
mptSetLayerColoringMethod(
     { d_cellViewID | d_techID }
     l_layers
     t_coloringMethod
     )
     => t / nil
```

Description

Sets the default coloring method for the technology database, the design, or specified layers in the design.

Arguments

d_cellViewID | *d_techID*	ID of the cellview or the technology database.
l_layers	List of layers for which the default coloring method will be set. If `nil`, then the default coloring method is set for the technology database (if *d_techID* was given) or the design (if *d_cellViewID* was given).
t_coloringMethod	Sets the default coloring method. If `nil`, the default coloring method is reset. Valid values: `"connectedShapes"`, `"colorSpacing"`, `nil`

Values Returned

`t`	The default coloring method was set or reset as requested.
`nil`	The default coloring method was not set.

Examples

```
mptSetLayerColoringMethod( geGetEditCellView() nil "colorSpacing" )
```

Sets the default coloring method for the current design to `colorSpacing`.

```
mptSetLayerColoringMethod( geGetEditCellView() list( "m1" ) "connectedShapes" )
```

Sets the default coloring method for layer `m1` of the current design to `connectedShapes`.

```
mptSetLayerColoringMethod( geGetEditCellView() nil nil )
```

Resets the default coloring method for the current design.

```
tfID = techGetTechFile( geGetEditCellView()->lib )
mptSetLayerColoringMethod( tfID nil "colorSpacing" )
```

Sets the default coloring method to `colorSpacing` for the technology database of the current cellview.

mptSetLayerDefaultColor

```
mptSetLayerDefaultColor(
     d_techFileID
     t_layerName
     t_color
     )
     => t / nil
```

Description

Sets the default color for a layer. This color will be assigned to newly created shapes and vias on the layer, and to gray shapes and vias on the layer when recoloring the design. The color will also be used to initialize the active color for this layer in the Palette.

Arguments

d_techFileID	Technology database ID.
t_layerName	Name of the layer.
t_color	The default color. Valid values: "nil", "grayColor", "mask<*mask number*>Color", "alternating", where <*mask number*> is 1, 2, and so on, up to the number of masks defined in the technology library for the layer.

Values Returned

t	The default color was set for the layer.
nil	The default color was not set for the layer.

Example

```
tfID=techGetTechFile( geGetEditCellView()~>lib )
mptSetLayerDefaultColor( tfID "m1" "mask1Color" )
mptSetLayerDefaultColor( tfID "m2" "mask2Color" )
```

Sets the default colors for layers m1 and m2 to mask1Color and mask2Color, respectively, for the current edit cellview.

mptSetLockDefaultColors

```
mptSetLockDefaultColors(
     g_lock
     )
     => t / nil
```

Description

Specifies whether the layer default color specified by mptSetLayerDefaultColor will be locked when assigned to a shape.

Arguments

g_lock	If `t`, lock the layer default color when it is assigned to a shape. If `nil`, do not lock the layer default color when it is assigned to a shape.

Values Returned

`t`	The lock state will be set as requested whenever the layer default color is assigned to a shape.
`nil`	The lock state will not be set as requested whenever the layer default color is assigned to a shape.

Example

```
tfID=techGetTechFile( geGetEditCellView()~>lib )
mptSetLayerDefaultColor( tfID "m1" "mask1Color" )
mptSetLockDefaultColors( t )
```

Locks the `mask1Color` color on layer `m1` whenever it is assigned as the layer default color to a shape.

mptSetTrackPatternPattern

```
mptSetTrackPatternPattern(
     d_trackPatternID
     l_colorPattern
     )
     => t / nil
```

Description

Specifies the color pattern for the track pattern.

Argument

d_trackPatternID	The database ID of the track pattern.
l_colorPattern	The list of color patterns to be set for the specified track pattern.

Values Returned

t	The color pattern is set for the track pattern.
nil	The color pattern is not set for the track pattern.

Example

```
cv =geGetEditCellView()
mptSetTrackPatternPattern( tp0 "GGRYB" )
    => t
```

Sets GGRYB as the color pattern for the track pattern, tp0.

mptSetupComplianceChecker

```
mptSetupComplianceChecker(
     dbObject
     [ x_startLevel ]
     [ x_stopLevel ]
     )
     => t / nil
```

Description

Flags any objects that are not aligned with the current MPT setup.

Arguments

dbObject	Database object of the cellview.
x_startLevel	Specifies the start level of query run to check compliance with the MPT setup.
x_stopLevel	Specifies the start level of query run to check compliance with the MPT setup.

Values Returned

`t`	The operation was successful and a report is displayed in the CIW.
`nil`	The operation was not successful.

Example

```
mptSetupComplianceChecker(cv)
```

mptShowMaskColor

```
mptShowMaskColor(
     t_layerName
     x_maskNumber
     g_toShow
     )
     => t / nil
```

Description

Shows or hides the color for the specified mask on the given layers.

Arguments

t_layerName	Name of the layer.
x_maskNumber	Mask number is 1, 2, and so on, up to the number of masks defined in the technology library for the layer.
g_toShow	If `t`, show the mask color. If `nil`, hide the mask color.

Values Returned

`t`	Shapes on the specified mask for the given layers are displayed with (show) or without (hide) the outline color for the mask.
`nil`	There was an error so the command was not completed.

Example

```
mptShowMaskColor( "Metal1" 2 nil )
```

Hides the color for mask2 on the `Metal1` layer.

mptUnlockAll

```
mptUnlockAll(
     d_cellViewID
     [ l_layerNames ]
     )
     => t / nil
```

Description

Unlocks all the colored shapes that are locked at the top level of the specified cellview. You can optionally unlock only the shapes on specific colorable layers. Top-level shapes with color attribute locks (dbLocks) are unlocked with their current color. Hierarchical color locks at the current editing hierarchy level are removed.

Arguments

d_cellViewID	The database ID of the cellview to be processed.
l_layerNames	Limits processing to the shapes on the colorable layers in this list.

Value Returned

`t`	The colored shapes are unlocked.
`nil`	The colored shapes are not unlocked due to an error (for example, invalid cellview ID or other invalid argument).

Example

```
mptUnlockAll( geGetEditCellView() )
```

Unlocks all the colored shapes at the current editing hierarchy level.

```
mptUnlockAll( cv list( "Metal1" "Metal2" ) )
```

Unlocks all the colored `Metal1` and `Metal2` top-level shapes of the `cv` cellview.

Related Topic

Lock and Unlock All

mptLockAll

mptUnpropagateLocks

```
mptUnpropagateLocks(
     d_cellViewID
     [ l_layerNames ]
     )
     => t / nil
```

Description

Removes the system locks created by the propagate locks command.

Arguments

db_ID	The database ID of the cellview to be processed.
l_layerNames	Limits processing to the shapes on the colorable layers in this list.

Value Returned

`t`	The command is successful.
`nil`	The command is not successful.

Example

```
mptUnpropagateLocks(deGetCellView(), "Metal2")
```

Removes all the system lock of the colored shapes of layer `Metal2`.

mptUpdateColor

```
mptUpdateColor(
     d_cellViewID
     [ l_layers ]
     [ g_depth ]
     )
     => t / nil
```

Description

Updates the color information from the bottom hierarchically, according to the specified depth and layers. This is useful when the color engine has been switched off/on, or a third party tool has been used to modify color information, leaving color information in an out-of-date state.

This function is similar to mptReColor but runs more quickly in cases when only some of the design is out of date, because valid color information is not recomputed. If the design has never been colored, this function works the same as mptReColor.

Arguments

d_cellViewID	The database ID of the top-level cellview where the color update is to be started and performed through the hierarchy.
l_layers	List of colorable layer names. If `nil`, all colorable layers are processed.
g_depth	Depth (hierarchy level) for bottom-up processing. If not specified, the entire hierarchy will be processed.

Values Returned

`t`	The color information update was successful.
`nil`	There was an error so the color information update was not completed.

Example

```
mptUpdateColor( geGetEditCellView() )
```

Updates the color for the hierarchy starting from the current top cell displayed in the current layout edit window.

```
mptUpdateColor( geGetEditCellView() list( "m1" "m2" ) )
```

Updates the color for the current edit cellview through the entire hierarchy on layers `m1` and `m2`.

```
mptUpdateColor( geGetEditCellView() nil 2 )
```

16

Advanced Boolean Engine Functions

This topic provides a list of basic editing Cadence® SKILL functions associated with Advanced Boolean Engine. These functions have the prefix `abe`.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Advanced Boolean Engine and its Layers

The Advanced Boolean Engine (ABE) is a set of geometric processing commands to perform layer operations such as AND, ANDNOT, OR, XOR, TOUCH, INSIDE, and OUTSIDE. ABE can also do SIZE, GROW, and SHRINK operations.

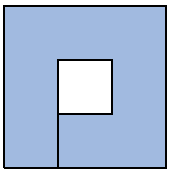
ABE supports multithreading to run multiple layer operations in parallel.

Here are a few limitations of ABE:

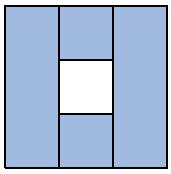
- ABE does not operate on all-angle geometries. Only octalinear shapes (90 and 45 degree angles) are supported.
- ABE layers are flat, non-hierarchical layers.
- ABE layers do not preserve logical connectivity.

ABE Layers

ABE operates on ABE layers that are temporary layers in memory, not layers in the cellview. ABE layers are mosaic-based structures that perform efficient operations by decomposing all objects into *tiles*, either rectangles or octagons.



Polygon



Polygon represented as four primitive tiles

Examples

To highlight a set, use this boolean operation:

```
; Initialize ABE
abeInit( geGetEditCellView() )
; Load input layers from cellview
M1 = abeLayerFromCellView( "Metal1" )
M2 = abeLayerFromCellView( "Metal2" ?purpose "drawing" ?depth 3)
; Create ABE output layers
m1Straddlem2 = abeNewLayer()
```

```
; ABE operation
abeLayerStraddle( M1 M2 m1Straddlem2 )
; Create a highlight set with the ABE output layer
hSet = geCreateWindowHilightSet( hiGetCurrentWindow() list( "Metal7" "drawing" ))
hSet->enable = t
abeLayerToHilightSet( m1Straddlem2 hSet )
; End the ABE session and release the memory
abeDone()
```

For cellview use this boolean operation:

```
; Initialize ABE
abeInit( geGetEditCellView() )
; Load input layers from cellview
M1 = abeLayerFromCellView( "Metal1" )
M2 = abeLayerFromCellView( "Metal2" ?purpose "drawing" )
; Create ABE output layers
m1Xorm2 = abeNewLayer()
; ABE operation
abeLayerXor( M1 M2 m1Xorm2 )
; Output the ABE Xor result to the Poly:drawing layer-purpose pair
abeLayerToCellView( m1Xorm2 "Poly" )
; End the ABE session and release the memory
abeDone()
```

Perform ABE Multithreading Using Queue

```
; Initialize with 8 threads
abeInit( geGetEditCellView() ?threads 8 )
; Load input layers from cellview
Poly = abeLayerFromCellView( "Poly" )
M1 = abeLayerFromCellView( "Metal1" )
M2 = abeLayerFromCellView( "Metal2" )
M3 = abeLayerFromCellView( "Metal3" )
M4 = abeLayerFromCellView( "Metal4" )
; Create ABE output layers
outLayer1 = abeNewLayer()
outLayer2 = abeNewLayer()
outLayer3 = abeNewLayer()
outLayer4 = abeNewLayer()
outLayer5 = abeNewLayer()
outLayer6 = abeNewLayer()
; ABE operations
abeLayerOr( Poly outLayer1 ?queue t )
abeLayerAnd( M1 M2 outLayer2 ?queue t )
abeLayerStraddle( M1 M2 outLayer3 ?queue t )
abeLayerSize( M2 outLayer4 0.25 ?queue t )
abeLayerGrow( M3 outLayer5 ?south 0.40 ?queue t )
abeLayerShrink( M4 outLayer6 ?east 0.20 ?west 0.22 ?queue t )
```

```
; Run the queue
abeRunQueue()
; Create a highlight set with an ABE output layer
hSet = geCreateWindowHilightSet( hiGetCurrentWindow() list( "Metal7" "drawing" ))
hSet->enable = t
abeLayerToHilightSet( outLayer2 hSet )
; End the ABE session and release the memory
abeDone()
```

Related Topics

Support for Multithreading in ABE Functions

Operating ABE

To operate ABE and its functions:

1. Use `abeInit` to start an ABE session for a given cellview. ABE functions can only be performed after ABE is initialized.
2. Create ABE layers as a clean layer or populated with shapes from a cellview.

   Hierarchical shapes can be copied from a cellview but are flattened on the ABE layer. Shapes can also be added to an ABE layer using a list of points.
3. To perform two-layer ABE operations you need two input ABE layers and have the general format:

   ```
   abeLayerOp(
   o_abeLayer1
   o_abeLayer2
   o_abeOutputLayer
   )
   =>t / nil
   ```

   One-layer ABE operations require one ABE input layer and have the following general format:

   ```
   abeLayerOp(
   o_abeInputLayer
   o_abeOutputLayer
   )
   =>t / nil
   ```
4. Save ABE layers to the current ABE session cellview or a Graphics Editor highlight set.
5. Use `abeDone` to end an ABE session. This frees up the memory that was used for the session.
6. Use `abeClearLayer` to clear an ABE layer. This is helpful to re-use existing ABE layers.
7. Use `abeRemoveLayer` to remove an ABE layer. This frees up the memory that was used for the ABE layer and is useful when you are doing a series of operations. For large designs, a significant amount of memory can be used by each ABE operation.

Related Topics

abeInit

abeLayerOrPtArray

abeDone

Examination of ABE Layers

ABE layers are not directly viewable in the canvas. ABE layer contents are copied to a Graphics Editor highlight set, from which they can be viewed in the canvas. Additionally, an ABE layer can be queried for the number of tiles on the layer and iterators can be used to identify the location of islands and tiles.

These properties are used for the ABE layer:

- The `area` property for the ABE layer contains the total area of all the tiles in the ABE layer.

  The following example copies the shapes in the selected set to ABE layer `a1`, then outputs the number of tiles on `a1`.

  ```
  abeInit( cv )
  a1 = abeLayerFromShapes( geGetSelSet() )
  print( a1->area )
  ```

- The `numTiles` property for the ABE layer contains the number of tiles on the layer.

  The following example copies the shapes in the selected set to ABE layer `a1`, then outputs the number of tiles on `a1`.

  ```
  abeInit( cv )
  a1 = abeLayerFromShapes( geGetSelSet() )
  print( a1->numTiles )
  ```

  If `numTiles` is `0`, the layer is empty.

These iterators are used for the ABE layer:

- The island iterator returns polygons and rectangles on an ABE layer, with each represented by a list of points. Use `abeIslandIterator` to set up an island iterator for an ABE layer. The `?noHoles` argument will generate polygons that do not contain holes.

  The following example copies the shapes in the selected set to ABE layer `a1`, then iterates through the polygons and rectangles to output the list of points for each shape.

  ```
  abeInit( cv )
  a1 = abeLayerFromShapes( geGetSelSet() )
  islandItr = abeIslandIterator( a1 )
  while( polygon = islandItr->next
     print( polygon )
  )
  ```

- The tile iterator returns the primitive tiles on an ABE layer. The iterator can be restricted to a region of the layer.

  The following example copies the shapes in the selected set to ABE layer `a1`, then iterates through the tiles to output the bounding box for each tile.

```
abeInit( cv )
a1 = abeLayerFromShapes( geGetSelSet() )
tileItr = abeTileIterator( a1 )
while( tile = tileItr->next
   print( tile->bbox )
)
```

Related Topics

abeIslandIterator

Support for Multithreading in ABE Functions

To apply multithreading to the ABE functions, you have to use the `?queue` argument. When `?queue` is specified, the function is queued for multithreaded processing which occurs only on the same host.

You can only apply the `?queue` argument to the two-layer ABE functions, one-layer ABE functions, and `abeLayerFromShapes`.

ABE limits the number of threads for multithreading to `(2*numberOfCores)-1)`. Therefore, for a two-core host, ABE would use at most three (3) threads. A larger number of threads could be requested but only the maximum number according to the formula would be used.

Limitations of multithreading:

- Within the same queue, the output of one ABE operation cannot be used as the input of another ABE operation.
- With the exception of `abeLayerGrow` and `abeLayerShrink`, multiple ABE operations can output to the same ABE layer.

There are three differences when running ABE with multithreading:

- When ABE is initialized, the number of threads can be specified as an argument to `abeInit`. The default is four threads.
- For the two-layer and one-layer ABE functions, add the `?queue` argument. This will push the operation onto a queue for processing. The operation is not run immediately, it is just added to a queue.
- Run the queue using `abeRunQueue`. All queued ABE operations run, and a return is issued after all have been completed.

Related Topics

Operating ABE

abeLayerFromShapes

abeLayerGrow

abeLayerShrink

abeInit

abeRunQueue

abeBlockagesFromCellView

```
abeBlockagesFromCellView(
     t_layerName
     o_outLayer
     g_queue
     [ ?maskColor g_maskColor ]
     )
     => o_abeLayer / nil
```

Description

Copies blockages from the specified layer in the current ABE session cellview to a new or existing ABE layer. Optional arguments restrict the copied blockages to the specified ABE layer. This function can be run immediately or added to the queue.

Arguments

t_layerName	Name of the layer in the technology database.
o_outLayer	ID of an existing ABE layer to which the cellview layer shapes will be added. This argument is required if `?queue` is specified.
g_queue	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.
`?maskColor` *g_maskColor*	Copies only the blockages from the cellview with the specified mask color. Valid values are: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, and `"mask4Color"`. The default is `"grayColor"`.

Value Returned

o_abeLayer	ID of the new ABE layer to which the layer shapes were added.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

The following example initializes the ABE engine with `cv (cellview id)`, then calls `abeBlockagesFromCellView` to copy all the blockages, that belong to `"Metal1"`, to the newly created ABE layer `'out1'`.

```
abeInit(cv)
out1 = abeBlockagesFromCellView("Metal1")
```

Related Topics

abeRunQueue

abeBoundariesFromCellView

```
abeBoundariesFromCellView(
     o_outLayer
     g_queue
     )
     => o_abeLayer / nil
```

Description

Copies the boundaries from the current ABE session cellview to a new ABE layer or, optionally, an existing ABE layer. Optional arguments restrict the copied boundaries to a specified layer. This function can be run immediately or added to the queue.

Arguments

`o_outLayer`	ID of an existing ABE layer to which the cellview layer shapes will be added. This argument is required if `?queue` is specified.
`g_queue`	Adds the function to the queue, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Value Returned

`o_abeLayer`	ID of the new ABE layer to which the layer shapes were added.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Example

The following example initializes the ABE engine with `cv (cellview id)` then calls `abeBoundariesFromCellView` to copy the boundaries to the newly created ABE layer out1.

```
abeInit(cv)
out1 = abeBoundariesFromCellView()
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeClearLayer

```
abeClearLayer(
     o_abeLayer
     )
     => t / nil
```

Description

Removes all the shapes on the ABE layer and frees the memory. This is useful for re-using ABE layers.

Arguments

o_abeLayer	ID of the ABE layer to be cleared.

Values Returned

`t`	The ABE layer is cleared of all shapes and the related memory is freed.
`nil`	The specified ABE layer could not be cleared.

Example

Creates and then clears the `andLayer` ABE layer.

```
andLayer = abeCreateLayer()
abeClearLayer( andLayer )
```

Related Topics

abeDone

abeDone

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

Description

Closes the current ABE session and frees all the memory that was used by ABE.

Arguments

None

Values Returned

`t`	The current ABE session is closed and all related memory is freed.
`nil`	There is no active ABE session.

Related Topics

abeClearLayer

abeElapsedTime

```
abeElapsedTime(
     [ ?reset g_reset ]
     )
     => g_time
```

Description

Returns the elapsed time for the current ABE session.

Arguments

`?reset g_reset`	Resets the clock to zero if set to `t`. By default, the clock is not reset.

Values Returned

`g_time`	Elapsed time in seconds.

Example

Returns the elapsed time.

```
abeElapsedTime()
```

Returns the elapsed time, then resets the elapsed time clock to zero.

```
abeElapsedTime( ?reset t )
```

Related Topics

Operating ABE

abeInit

```
abeInit(
     d_cellViewId
     [ ?doInterrupts g_interrupts ]
     [ ?depth g_depth ]
     [ ?threads g_threadCount ]
     )
     => t / nil
```

Description

Initializes an ABE session for the specified cellview. The ABE elapsed time clock is set to zero.

Arguments

d_cellViewId	ID of the cellview.
?doInterrupts *g_interrupts*	Allows ABE operations to be terminated using `control+c`.
?depth *g_depth*	Sets the hierarchy depth for loading shapes. If 0 is specified, only the shapes on the top level are processed. By default, all levels of the hierarchy are loaded to a flat structure.
?threads *g_threadCount*	The number of threads to use for ABE processing. By default, four threads are used, all on the same host. The maximum number of threads that can be used is based on the number of cores on the host `((2* numberOfCores)-1)`. If greater than the maximum number is specified, the maximum number of threads will be used.

Values Returned

`t`	ABE session is initialized for the given cellview.
`nil`	The ABE session was not initialized.

Example

Initializes an ABE session for the current edit cellview.

```
abeInit( geGetEditCellView() )
```

Related Topics

Operating ABE

abeIslandIterator

```
abeIslandIterator(
     o_abeLayer
     [ ?noHoles g_noHoles ]
     )
     => o_islandIter / nil
```

Description

Returns the island iterator for the rectangles and polygons on the ABE layer. Each entry is a list of points for a shape on the ABE layer.

Arguments

o_abeLayer	ID of the ABE layer.
?noHoles *g_noHoles*	Removes holes in shapes if set to `t`, so that the resulting shape is represented by the bounding polygon of the shape.

Values Returned

o_islandIter	ID of the ABE layer island iterator.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Gets the island iterator (`ii`) for ABE layer `a1`, then iterates through the list of points for each island on the layer.

```
ii = abeIslandIterator( a1 )
while( polygon = ii->next
    print( polygon )
)
```

Related Topics

Examination of ABE Layers

abeLayerAbut

```
abeLayerAbut(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue]
     )
     => t / nil
```

Description

Adds shapes to the ABE output layer for ABE layer1 shapes that abut with shapes on ABE layer2.

Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	Adds shapes to this ABE layer and the ABE layer created by this function.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes from ABE layer `a1` that abut with shapes from ABE layer `a2` to ABE layer `a3`.

```
abeLayerAbut( a1 a2 a3 )
```

Related Topics

abeRunQueue

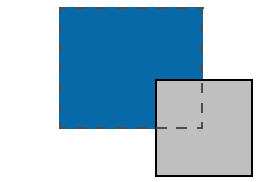
Support for Multithreading in ABE Functions

abeLayerAnd

```
abeLayerAnd(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

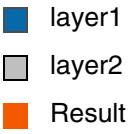
Description

Adds shapes to the ABE output layer where shapes on ABE layer1 overlap shapes on ABE layer2.



Original
layer1 and layer2 shapes

Resulting shapes
abeLayerAnd



Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	Adds shapes to this ABE layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function was run successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes to the `a3` ABE layer where `a1` ABE layer shapes overlap the shapes on the `a2` ABE layer.

```
abeLayerAnd( a1 a2 a3 )
```

Related Topics

abeRunQueue

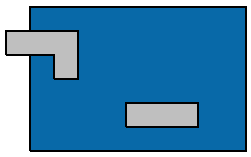
Support for Multithreading in ABE Functions

abeLayerAndNot

```
abeLayerAndNot(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

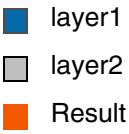
Description

Adds shapes to the ABE output layer where shapes on layer1 do not overlap shapes on layer2.



Original
layer1 and layer2 shapes

Resulting shapes
abeLayerAndNot



Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	Adds shapes to this ABE layer
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes to the `result` ABE layer where `a1` ABE layer shapes do not overlap shapes on the `a2` ABE layer.

```
abeLayerAndNot( a1 a2 result )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerAvoid

```
abeLayerAvoid(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     [ ?islandBased g_islandBased ]
     )
     => t / nil
```

Description

Adds shapes on the output layer for shapes on layer1 that are completely outside of, and have no abutting edges with, shapes on layer2.

Original
layer1 and layer2 shapes

Resulting shapes
abeLayerAvoid

Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	Adds shapes to the ABE layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.
?islandBased *g_islandBased*	Includes all the full islands that are not touching the ABE layer2.

Values Returned

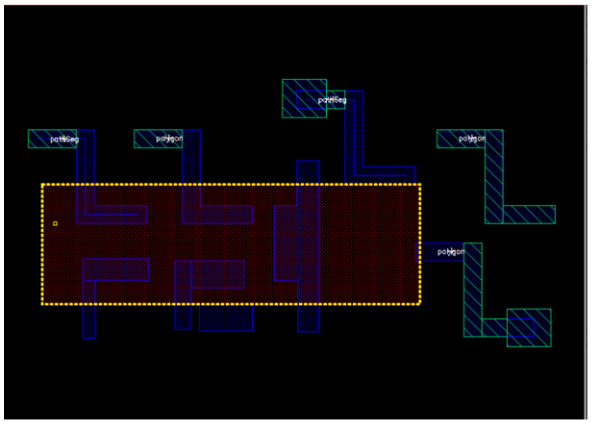
`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

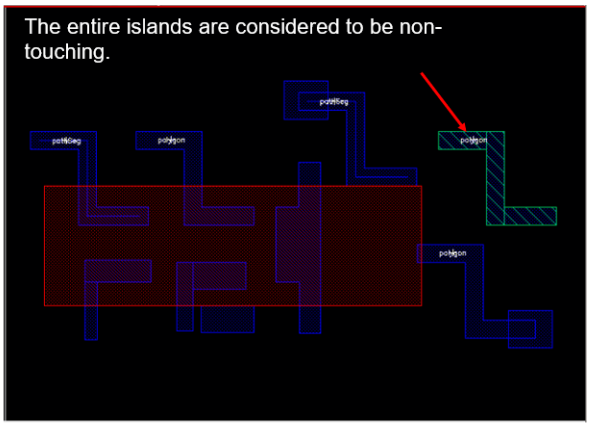
Adds shapes to the `result` ABE layer where `a1` ABE layer shapes are completely outside of, and have no abutting edges with, shapes on the `a2` ABE layer.

```
abeLayerAvoid( a1 a2 result )
```

When `?islandBased` is specified, it includes entire islands to the `outputLayer` ABE layer where `m1layer` ABE layer islands are completely outside and not touching the `m2layer` ABE layer. The shapes with green outlines and diagonal stripes are the resulting shapes from the ABE layer.



abeLayerAvoid(m1layer, m2layer outputLayer)



abeLayerAvoid(m1layer, m2layer outputLayer ?islandBased t)

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerFromCellView

```
abeLayerFromCellView(
     t_layerName
     [ ?bbox l_bbox ]
     [ ?bBox l_bBox ]
     [ ?purpose t_purposeName ]
     [ ?excludePurpose l_purposeName ]
     [ ?outLayer o_abeOutputLayer [ ?queue g_queue ] ]
     [ ?maskColor g_maskColor ]
     [ ?startLevel g_startLevel ]
     [ ?stopLevel g_stopLevel ]
     [ ?orientation g_orientation ]
     [ ?onlyPinFig g_onlyPinFig ]
     [ ?onlyVia g_onlyVia ]
     )
     => o_abeLayer / nil
```

Description

Copies the shapes from the specified layer in the current ABE session cellview to a new ABE layer or, optionally, an existing ABE layer. Optional arguments restrict the copied shapes to a specific area, purpose, mask color, or hierarchical depth. This function can be run immediately or added to the queue.

Arguments

t_layerName	Name of the layer in the technology database.
?bbox *l_bbox*	Processes only the shapes in the specified region, represented by a list of two window coordinates for the lower-left and the upper-right corner of the region. For example, `list(20:20 220:280)` Cadence recommends that you use `?bBox` instead of `?bbox` to specify this value.

`?bBox` *`l_bBox`*
: Processes only the shapes in the specified region, represented by a list of two window coordinates for the lower-left and the upper-right corner of the region. For example, `list(20:20 220:280)`

  Cadence recommends that you use `?bBox` instead of `?bbox` to specify this value.

  If both `?bbox` and `?bBox` are specified, value specified for `?bBox` is considered.

`?purpose` *`t_purposeName`*
: Loads only shapes from the specified purpose. By default, shapes from all purposes on the layer are loaded.

`?excludePurpose` *`l_purposeName`*
: Specifies provide list of purpose names to exclude.

`?outLayer` *`o_abeOutputLayer`*
: ID of an existing ABE layer to which the cellview layer shapes will be added. This argument is required if `?queue` is specified.

`?queue` *`g_queue`*
: Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

`?maskColor` *`g_maskColor`*
: Copies only the shapes with the specified mask color. Valid values are: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, and `"mask4Color"`. The default is `"grayColor"`.

`?startLevel` *`g_startLevel`*
: Specifies the start hierarchy level from which to copy the shapes. The default is `0` (top level).

`?stopLevel` *`g_stopLevel`*
: Specifies the stop hierarchy level from which to copy the shapes. The default is `32`.

`?orientation` *`g_orientation`*
: Specifies the orientation of shapes.

  Valid values: `horizontal` and `vertical`

`?onlyPinFig` *`g_onlyPinFig`*

Specifies when the ABE layer should be created with pin figures on the specified layers. These pin figures are via shapes and shapes associated with pins.

`?onlyVia` *`g_onlyVia`* Specifies when the ABE layer should be created with via shapes on the specified layer.

Values Returned

`o_abeLayer` ID of the new ABE layer to which the layer shapes were added.

Via shapes associated with pins are added to the output layer only when the `onlyVia` and `onlyPinFig` arguments are set to `t`.

`nil` The function was not successful because ABE is not initialized or there was an argument error.

Examples

Creates a new ABE layer `a2` with shapes from the layer `Metal2`.

```
a2 = abeLayerFromCellView("Metal2")
```

Creates a new ABE layer `a2` with shapes from the layer-purpose pair `Metal2:drawing`.

```
a2 = abeLayerFromCellView("Metal2" ?purpose "drawing")
```

Creates a new ABE layer `a2` with only the `mask1Color` shapes copied from the layer-purpose pair `Metal2:drawing`.

```
a2 = abeLayerFromCellView("Metal2" ?purpose "drawing" ?maskColor "mask1Color")
```

Creates a new ABE layer `a2` with only the `Metal2` shapes `at` levels 1 and 2.

```
a2 = abeLayerFromCellView("Metal2" ?startLevel 1 ?stopLevel 2)
```

Creates a new ABE layer `a2` to get pin figures on the layer `Metal1`.

```
a2 = abeLayerFromCellView("Metal1" ?onlyPinFig t)
```

Creates a new ABE layer `a2` to get via shapes on the layer `Metal1`.

```
a2 = abeLayerFromCellView("Metal1" ?onlyVia t)
```

Creates a new ABE layer `a2` to get via that are pin figures on the layer `Metal1`.

```
a2 = abeLayerFromCellView("Metal1" ?onlyPinFig t ?onlyVia t)
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerFromFigSet

```
abeLayerFromFigSet(
     t_figSet
     [ ?outLayer o_abeOutputLayer ]
     )
     => o_abeLayer / nil
```

Description

Adds the shapes from a FigSet to an existing ABE layer or a new ABE layer.

Arguments

t_figSet	Name of the FigSet.
?outLayer *o_abeOutputLayer*	ID of the ABE layer to which the shapes from the FigSet will be added. If this argument is not included, a new ABE layer is created.

Values Returned

o_abeLayer	ID of the new or existing ABE layer to which the FigSet shapes were added.
nil	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Creates ABE layer `af1` that contains shapes from the `figset1` FigSet.

```
cv = geGetEditCellView()
f1 = txCreateFigSet( cv "figset1" )
txAppendObjectToFigSet( f1 dbCreateRect( cv '(0 "drawing") '((0 0) (5 7)) ) )
af1 = abeLayerFromFigSet( "figset1" )
```

Related Topics

abeLayerFromShapes

abeLayerFromNet

```
abeLayerFromNet(
     t_layerName
     t_netName
     [ ?outLayer o_abeOutputLayer ]
     [ ?onlyPinFig g_onlyPinFig ]
     [ ?onlyVia g_onlyVia ]
     [ ?maskColor g_maskColor ]
     )
     => o_abeLayer / nil
```

Description

Adds the shapes from a net to an existing or new ABE layer. Logical connectivity is not preserved and therefore cannot be restored if the shapes are written back to the ABE session cellview.

Arguments

t_layerName	Name of the layer in the technology database.
t_netName	Name of the net.
?outLayer *o_abeOutputLayer*	Specifies the ID of the ABE layer to which the shapes from the net will be added. If this argument is not included, a new ABE layer is created.
?onlyPinFig *g_onlyPinFig*	Specifies when the ABE layer should be created with pin figures from the net on the specified layer. These pin figures are via shapes and shapes associated with pins.
?onlyVia *g_onlyVia*	Specifies when the ABE layer should be created with via shapes from the net on the specified layer.
?maskColor *g_maskColor*	Copies only the shapes from the net with the specified mask color. Valid values are: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, and `"mask4Color"`. The default is `"grayColor"`.

Values Returned

o_abeLayer	ID of the new or existing ABE layer to which the net shapes were added. Via shapes associated with pins are added to the output layer only when the `?onlyVia` and `?onlyPinFig` arguments are set to `t`.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Creates the ABE layer `a1` that contains the `Metal1` shapes for `NetA`.

```
a1 = abeLayerFromNet( "Metal1" "NetA" )
```

Creates the ABE layer `a1` to get pin figures from `net1` on the layer `Metal1`.

```
a1 = abeLayerFromNet( "Metal1" "net1" ?onlyPinFig t )
```

Creates the ABE layer `a1` to get via shapes from `net1` on the layer `Metal1`.

```
a1 = abeLayerFromNet( "Metal1" "net1" ?onlyVia t )
```

Creates the ABE layer `a1` to get vias that are pin figures from `net1` on the layer `Metal1`.

```
a1 = abeLayerFromNet( "Metal1" "net1" ?onlyPinFig t ?onlyVia t )
```

Related Topics

abeLayerFromShapes

abeLayerFromShapes

```
abeLayerFromShapes(
     l_shapeObj
     [ ?maskColor g_maskColor ]

     )
     => o_abeLayer / nil
```

Description

Creates an ABE layer with shapes from the given list.

Arguments

l_shapeObj	Lists the shape objects.
?maskColor *g_maskColor*	Copies only the shapes with the specified mask color. Valid values are: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, and `"mask4Color"`. The default is `"grayColor"`.

Values Returned

o_abeLayer	ID for the new ABE layer with the copied shapes.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Creates ABE layer `a5` that contains the shapes in the selected set.

```
a5 = abeLayerFromShapes( geGetSelSet() )
```

Related Topics

abeLayerFromNet

abeLayerGrow

```
abeLayerGrow(
     o_abeInputLayer
     o_abeOutputLayer
     [ ?north g_sizeNorth ]
     [ ?south g_sizeSouth ]
     [ ?east g_sizeEast ]
     [ ?west g_sizeWest]
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Adds shapes to the ABE output layer that are ABE input layer shapes that have been increased in size along one or more directions in the north, south, east, and/or west.

Original
layer1 shapes

Resulting shapes
abeLayerGrow (all directions equally)

Arguments

o_abeInputLayer	ID of the ABE input layer.
o_abeOutputLayer	ID of the ABE output layer.
?north *g_sizeNorth*	Expands the north side of the shapes by this distance, in user units. The default value is 0.
?south *g_sizeSouth*	Expands the south side of the shapes by this distance, in user units. The default value is 0.
?east *g_sizeEast*	Expands the east side of the shapes by this distance, in user units. The default value is 0.
?west *g_sizeWest*	Expands the west side of the shapes by this distance, in user units. The default value is 0.

`?queue` *`g_queue`*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The expanded shapes are added on the output layer or the function is queued.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes from the selected set that have been increased in size by 0.02 to the north and 0.05 to the east to the `result` ABE layer.

```
a1 = abeLayerFromShapes( geGetSelSet() )
result = abeNewLayer()
abeLayerGrow( a1 result ?north 0.02 ?east 0.05 )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerInside

```
abeLayerInside(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Adds ABE layer1 shapes that are completely inside an ABE layer2 shape to the ABE output layer. Coincident edges are considered inside. Partially overlapping shapes are excluded.

Original
layer1 and layer2 shapes

Resulting shapes
abeLayerInside

Set the environment variable `abeFixShapePartitioning` to enable the proper functioning of non-rectangular shapes.

Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	ID of the ABE output layer.
?queue *g_queue*	Adds the function to the queue, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds `a1` ABE layer shapes that are completely inside an `a2` ABE layer shape to the `result` ABE layer.

```
abeLayerInside( a1 a2 result )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeFixShapePartitioning

abeLayerMergeTiles

```
abeLayerMergeTiles(
     i_abeLayer
     t_orientation
     [ ?queue { t | nil } ]
     => o_outputLayer / nil
```

Description

Uses an existing ABE layer to recreate maximum tiles based on the specified orientation.

Arguments

i_abeLayer	A valid ABE layer.
t_orientation	The orientation of shapes. Valid values are `horizontal` and `vertical`.
`?queue`	Adds the function to the queue, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Value Returned

o_outputLayer	ID of the ABE output layer.
`nil`	The operation failed.

Examples

```
cv = dbOpenCellViewByType("Test" "cmdTest" "layout" "maskLayout" "w")
abeInit(cv)
l = abeLayerFromCellView("Metal1")
```

Here ABE layer `l` has tiles generated by default. If you want the maximum possible number of tiles to be vertical, run the following:

```
abeLayerMergeTiles(l "vertical")
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerOr

```
abeLayerOr(
     o_abeInputLayer
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Adds shapes from the input ABE layer to the ABE output layer.

Original
layer1 and layer2 shapes

Resulting shapes
Or

Arguments

o_abeInputLayer	ID of the ABE input layer.
o_abeOutputLayer	ID of the ABE output layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds `a1` ABE layer shapes to the `result` ABE layer.

```
abeLayerOr( a1 result )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerOrPtArray

```
abeLayerOrPtArray(
     l_pointArray
     o_abeOutputLayer
     )
     => t / nil
```

Description

Adds a polygon that is represented by a list of points to the ABE output layer.

Arguments

l_pointArray	List of points representing the polygon to be added.
o_abeOutputLayer	ID of the ABE output layer.

Values Returned

`t`	Adds the polygon to the ABE output layer.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds the polygon represented by the `pts` point array to ABE layer `a7`.

```
abeLayerOrPtArray( pts a7 )
```

Related Topics

abeLayerOr

abeLayerOutside

```
abeLayerOutside(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Adds to the ABE output layer the ABE layer1 shapes that are completely outside the ABE layer2 shapes. Coincident edges are considered outside. Partially overlapping shapes are excluded.

Original
layer1 and layer2 shapes

Resulting shapes
outside

Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	ID of the ABE output layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function is run successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds `a1` ABE layer shapes that are completely outside an `a2` ABE layer shape to the `result` ABE layer.

```
abeLayerOutside( a1 a2 result )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

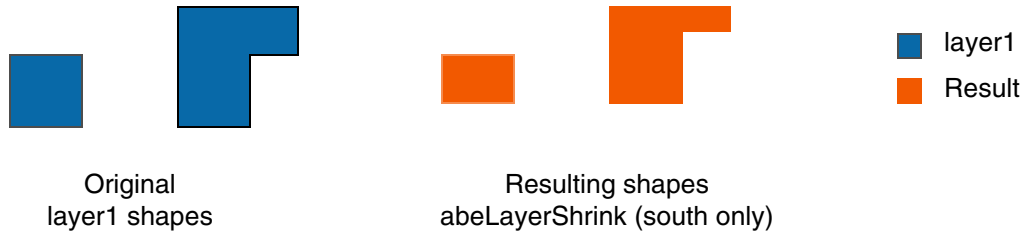
abeLayerInside

abeLayerShrink

```
abeLayerShrink(
     o_abeInputLayer
     o_abeOutputLayer
     [ ?north g_sizeNorth ]
     [ ?south g_sizeSouth ]
     [ ?east g_sizeEast ]
     [ ?west g_sizeWest]
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Decreases the size of shapes from the input ABE layer and outputs them to the ABE output layer. Shapes can be reduced by different distances in the north, south, east, and west directions.



Arguments

o_abeInputLayer	ID for the ABE input layer.
o_abeOutputLayer	ID for the ABE output layer.
?north *g_sizeNorth*	Shrinks the north side of the shapes by this distance, in user units. The default value is 0.
?south *g_sizeSouth*	Shrinks the south side of the shapes by this distance, in user units. The default value is 0.
?east *g_sizeEast*	Shrinks the east side of the shapes by this distance, in user units. The default value is 0.
?west *g_sizeWest*	Shrinks the west side of the shapes by this distance, in user units. The default value is 0.

`?queue` *`g_queue`*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The reduced shapes are added to the output layer or the function was queued.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes from the `a1` ABE layer that have been decreased in size by 0.05 on the south side to the `result` ABE layer.

```
abeLayerShrink( a1 result ?south 0.05)
```

Related Topics

abeRunQueue

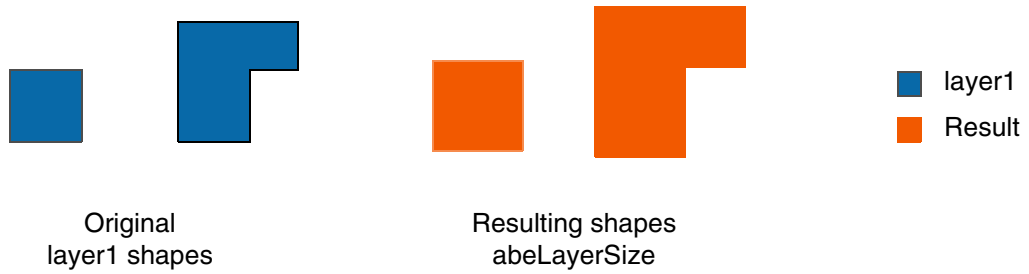
Support for Multithreading in ABE Functions

abeLayerSize

```
abeLayerSize(
     o_abeInputLayer
     o_abeOutputLayer
     g_size
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Adds shapes that have been increased or decreased in size from the ABE input layer to the ABE output layer.



Original
layer1 shapes

Resulting shapes
abeLayerSize

Arguments

o_abeInputLayer	ID for the ABE input layer.
o_abeOutputLayer	ID for the ABE output layer.
g_size	Expands or shrinks the size of shapes by this distance, in user units. A positive value increases the size; a negative value decreases the size.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	New shapes are created on the ABE output layer or the function was queued.

`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes that have been expanded by 0.05 in all directions from the `a1` ABE layer to the `result` ABE layer.

```
abeLayerSize( a1 result 0.05 )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerStraddle

```
abeLayerStraddle(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

Description

Adds the ABE layer1 shapes that partially overlap an ABE layer2 shape to the ABE output layer.

Original
layer1 and layer2 shapes

Resulting shapes
abeLayerStraddle

Arguments

o_abeLayer1	ID for the temporary ABE layer1.
o_abeLayer2	ID for the temporary ABE layer2.
o_abeOutputLayer	ID for the temporary ABE output layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function is run successfully.

`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds `a1` ABE layer shapes that partially overlap an `a2` ABE layer shape to the `result` ABE layer.

```
abeLayerStraddle( a1 a2 result)
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeLayerToBlockages

```
abeLayerToBlockages(
     o_abeInputLayer
     t_layerName
     [ ?maskColor g_maskColor ]
     [ ?blockageType g_blockageType ]
     [ ?tiles g_tiles ]
     [ ?noHoles g_noHoles ]
     [ ?blockageIds g_blockageIds ]
     )
     => t / nil
```

Description

Adds shapes as blockage objects from the ABE layer to the specified layer. Optional arguments specify a mask color and blockage type for the blockage objects, whether the shapes are copied without holes, and whether shapes are copied as tiles instead of islands.

Arguments

o_abeInputLayer	ID of the ABE input layer.
t_layerName	Name of the layer to which the ABE layer shapes will be copied as blockage objects.
?maskColor *g_maskColor*	Assigns the specified mask color to the new blockage objects. Valid values are: "grayColor", "mask1Color", "mask2Color", "mask3Color", and "mask4Color". The default is "grayColor".
?blockageType *g_blockageType*	Specifies the blockage type for the blockage objects. Valid values are: "routing", "via", "wiring", "fill", "slot", "pin", "feedthru", and "screen". The default is "routing".
?tiles *g_tiles*	Copies the shapes from the ABE layer as primitive tile blockage shapes when set to t. By default (nil), shapes are copied as islands. ?tiles and ?noHoles are mutually exclusive. If both are specified, the shapes are copied as primitive tiles with holes.

`?noHoles` *`g_noHoles`*	Covers holes in the added shapes. By default, holes are not filled in. `?tiles` and `?noHoles` are mutually exclusive. If both are specified, the shapes are copied as primitive tiles with holes.
`?blockageIds` *`g_blockageIds`*	Returns list of blockage IDs in the specified ABE layer.

Values Returned

`l_blockageIDs`	Lists blockage IDs when the `?blockageIds` argument is set to `t`.
`t`	The ABE layer shapes were added to the specified layer as blockage objects.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Copies the shapes on the ABE layer `m1` as blockage objects of type `routing` to `Metal2`.

```
abeLayerToBlockages(m1 "Metal2")
```

Related Topics

abeBlockagesFromCellView

abeLayerToCellView

```
abeLayerToCellView(
     o_abeLayer
     t_layerName
     [ ?tiles g_tiles
     | ?noHoles g_noHoles ]
     [ ?purpose t_purposeName ]
     [ ?maskColor g_maskColor ]
     [ ?cell t_cellName ]
     [ ?view t_viewName ]
     [ ?cvId d_cellviewID ]
     [ ?shapeIds { t | nil} }
     )
     => l_shapeIDs / t / nil
```

Description

Copies shapes from the ABE layer to the specified layer in the current ABE session cellview. Optional arguments specify the layer purpose, whether the shapes are copied without holes, and whether the shapes are copied as islands instead of primitive tiles.

Arguments

o_abeOutputLayer	ID of the ABE layer.
t_layerName	Displays the name of the layer to which the ABE layer shapes will be copied.
?tiles *g_tiles*	Copies the shapes from the ABE layer as primitive tiles when set to `t` (default). `?tiles` and `?noHoles` are mutually exclusive. If both are specified, the shapes are copied as primitive tiles with holes.
?noHoles *g_noHoles*	Fills holes in the added shapes when set to `t`. By default, holes are not filled in. `?tiles` and `?noHoles` are mutually exclusive. If both are specified, the shapes are copied as primitive tiles with holes.
?blockageType *g_blockageType*	Displays the name of the layer purpose for the copied shapes. The purpose must be in the technology database. The default purpose is `drawing`.
?maskColor *g_maskColor*	

	Assigns the specified mask color to the copied shapes. Valid values are: `"grayColor"`, `"mask1Color"`, `"mask2Color"`, `"mask3Color"`, and `"mask4Color"`. The default is `"grayColor"`.
`?cell` *`t_cellName`*	Specifies the cell name of a cellview. This option is required if `?view` is specified.
`?view` *`t_viewName`*	Specifies the view name of a cellview. This option is required if `?cell` is specified.
`?cvId` *`d_cellviewID`*	Database ID of a cellview. You should either specify cellview by providing cell and view name (using arguments `?cell` and `?view`) or specify the `?cvId` argument. If both are provided, `?cvId` is considered.
`?shapeIds`	Returns list of shape IDs in the specified ABE layer.

Values Returned

`l_shapeIDs`	Lists shape IDs when the `?shapeIds` argument is set to `t`.
`t`	The ABE layer shapes were copied to the specified layer.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Copies the shapes from the `abe1` ABE layer to the `Metal6:drawing` layer-purpose pair.

```
abeLayerToCellView(abe1 "Metal6")
```

Copies the shapes from the `abe1` ABE layer to the `Metal6:dummy` layer-purpose pair.

```
abeLayerToCellView(abe1 "Metal6" ?purpose "dummy")
```

Copies the shapes from the `abe1` ABE layer to the `Metal6:drawing` layer-purpose pair with no holes in the copied shapes.

```
abeLayerToCellView(abe1 "Metal6" ?noHoles)
```

Copies the shapes as primitive tiles from the `abe1` ABE layer to the `Metal6:drawing` layer-purpose pair.

```
abeLayerToCellView(abe1 "Metal6" ?tiles)
```

Copies the shapes from the `abe1` ABE layer to the cellview corresponding to input cellview ID.

```
abeLayerToCellView( abe1 "Metal6" ?cvId cellViewId)
```

Copies the shapes from the `abe1` ABE layer to the cellview corresponding to the name as specified in the input.

```
abeLayerToCellView( abe1 "Metal6" ?cell "cellName" ?view "viewName")
```

Copies the shapes from the `abe1` ABE layer to the `Metal6:drawing` layer-purpose pair and returns the list of shape IDs.

```
abeLayerToCellView(abe1 "Metal6" ?shapeIds t)
```

Copies the shapes from the `abe1` ABE layer to the `Metal:drawing` layer-purpose pair and assigns the `mask2Color` to the copied shapes.

```
abeLayerToCellView(abe1 "Metal1" ?maskColor "mask2Color")
```

Related Topics

abeLayerFromShapes

abeLayerToHilightSet

```
abeLayerToHilightSet(
     o_abeInputLayer
     h_hilightSetId
     [ ?tiles g_tiles
     | ?noHoles g_noHoles ]
     )
     => t / nil
```

Description

Adds the rectangles and polygons on the specified ABE layer to the specified highlight set. Optional arguments specify whether the shapes are copied without holes, and whether the shapes are copied as primitive tiles instead of islands.

Arguments

o_abeInputLayer	ID of the ABE input layer.
h_hilightSetId	ID of the highlight set.
?tiles *g_tiles*	Copies the shapes from the ABE layer as primitive tiles. By default (`nil`), shapes are copied as islands. `?tiles` and `?noHoles` are mutually exclusive. If both are specified, the shapes are copied as primitive tiles with holes.
?noHoles *g_noHoles*	Fills holes in the added shapes. By default, holes are not filled in. `?tiles` and `?noHoles` are mutually exclusive. If both are specified, the shapes are copied as primitive tiles with holes.

Values Returned

`t`	Adds the shapes from the ABE layer to the highlight set.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds the `a2` ABE layer shapes to the highlight set `hSet`.

```
hSet = geCreateWindowHilightSet( hiGetCurrentWindow() list( "Metal6" "drawing"))
abeLayerToHilightSet( a2 hSet )
```

Related Topics

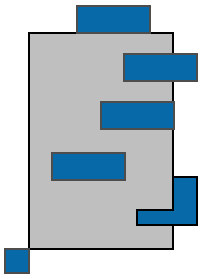
abeIslandIterator

abeLayerTouch

```
abeLayerTouch(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     [ ?includeCorners g_includeCorners ]
     [ ?islandBased g_islandBased ]
     )
     => t / nil
```

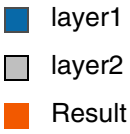
Description

Adds the ABE layer1 shapes that touch (completely overlap, partially overlap, or abut) an ABE layer2 shape to the ABE output layer.



Original
layer1 and layer2 shapes

Resulting shapes
abeLayerTouch



Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	ID of the ABE output layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.
?includeCorners *g_includeCorners*	Includes shapes that touch only at corners. The default is `nil`, which means corner-only touching shapes are not included.
?islandBased *g_islandBased*	Includes all the full islands that are touching the ABE layer1.

Values Returned

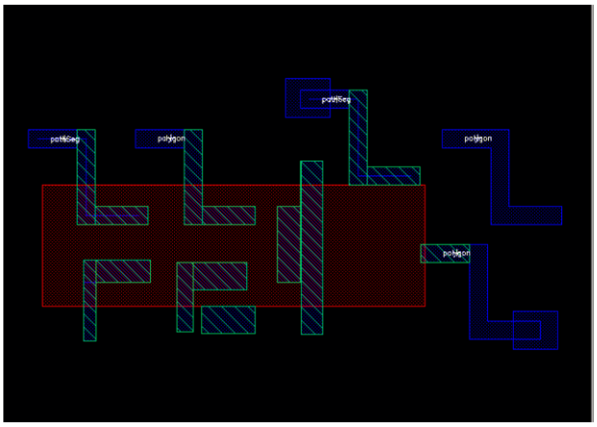
`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

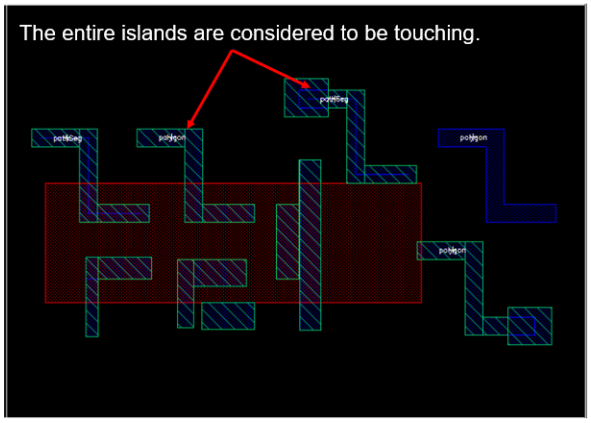
Adds shapes from the `a1` ABE layer that completely overlap, partially overlap, or abut an `a2` ABE layer shape to the `result` ABE layer.

```
abeLayerTouch( a1 a2 result )
```

When `?islandBased` is specified, it includes entire islands to the `outputLayer` ABE layer where `m1layer` ABE layer islands are touching the `m2layer` ABE layer. The shapes with green outlines and diagonal stripes are the resulting shapes from the ABE layer.



abeLayerTouch(m1layer, m2layer outputLayer)



abeLayerTouch(m1layer, m2layer outputLayer ?islandBased t)

Related Topics

abeRunQueue

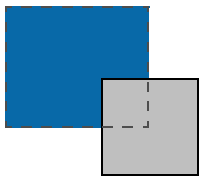
Support for Multithreading in ABE Functions

abeLayerXor

```
abeLayerXor(
     o_abeLayer1
     o_abeLayer2
     o_abeOutputLayer
     [ ?queue g_queue ]
     )
     => t / nil
```

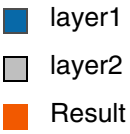
Description

Adds rectangles and polygons to the ABE output layer where ABE layer1 shapes and ABE layer2 shapes do not overlap.



Original
layer1 and layer2 shapes

Resulting shapes
abeLayerXor



Arguments

o_abeLayer1	ID of ABE layer1.
o_abeLayer2	ID of ABE layer2.
o_abeOutputLayer	ID of the ABE output layer.
?queue *g_queue*	Adds the function to the queue if set to `t`, rather than running it immediately. Functions in the queue are not run until `abeRunQueue` is issued.

Values Returned

`t`	The function was run or queued successfully.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Adds shapes to the `result` ABE layer where shapes on the `a1` ABE layer and the `a2` layer do not overlap.

```
abeLayerXor( a1 a2 result )
```

Related Topics

abeRunQueue

Support for Multithreading in ABE Functions

abeMTdebug

```
abeMTdebug(
     o_abeLayer
     )
     => t / nil
```

Description

Enables the output of additional messages to monitor progress when multithreading is used.

Arguments

o_abeLayer	ID of the ABE layer.

Values Returned

`t`	Multithreading message output is enabled.
`nil`	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Enables multithreading message output.

```
abeMTdebug( a1 )
```

Related Topics

Support for Multithreading in ABE Functions

abeNewLayer

```
abeNewLayer(
     )
     => o_abeLayer / nil
```

Description

Creates a new ABE layer.

Arguments

None

Values Returned

o_abelayer	ID of the newly created ABE layer.
nil	The function was not successful because ABE is not initialized.

Examples

Creates a new a1 ABE layer.

```
a1 = abeNewLayer()
```

Related Topics

Operating ABE

abeRemoveLayer

```
abeRemoveLayer(
     o_abeLayer
     )
     => t / nil
```

Description

Removes the specified ABE layer.

Arguments

o_abeLayer	ID of the ABE layer.

Values Returned

`t`	The ABE layer was removed.
`nil`	The function was not successful because ABE is not initialized or the ABE layer does not exist.

Examples

Removes the `a1` ABE layer and frees the memory associated with it.

```
abeRemoveLayer( a1 )
```

Related Topics

Operating ABE

abeRunQueue

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

Description

Runs all the ABE operations that are queued for processing.

Arguments

None

Values Returned

`t`	All ABE functions that were queued have been processed.
`nil`	The function was not successful because ABE is not initialized or the queue is empty.

Examples

Runs ABE functions in the queue.

```
abeRunQueue()
```

Related Topics

Support for Multithreading in ABE Functions

abeShapesInside

```
abeShapesInside(
     d_cellViewId
     l_insideShapeObj
     l_outsideShapeObj
     )
     => l_shapeObj / nil
```

Description

Returns a list of shapes from the inside list that are completely inside a shape from the outside list. Coincident edges are considered inside. Partially overlapping shapes are excluded.

Arguments

d_cellViewId	ID of the cellview.
l_insideShapeObj	List of inside shapes.
l_outsideShapeObj	List of outside shapes.

Values Returned

l_shapeObj	List of shapes that are completely inside a shape from the outside list.
nil	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Returns a list of shapes from the selected set in the current cellview that are completely inside a shape from the L2 list of shapes.

```
abeShapesInside( geGetEditCellView() geGetSelSet() L2 )
```

Related Topics

abeLayerInside

abeShapesOutside

```
abeShapesOutside(
     d_cellViewId
     l_outsideShapeObj
     l_insideShapeObj
     )
     => l_shapeObj / nil
```

Description

Returns a list of shapes from the outside list that are completely outside of shapes from the inside list. Coincident edges are considered outside. Partially overlapping shapes are excluded.

Arguments

d_cellViewId	ID of the cellview.
l_outsideShapeObj	List of outside shapes.
l_insideShapeObj	List of inside shapes.

Values Returned

l_shapeObj	List of shapes from the outside list that are completely outside a shape from the inside list.
nil	The function was not successful because ABE is not initialized or there was an argument error.

Examples

Returns a list of shapes from the selected set in the current cellview that are completely outside a shape from the L2 list of shapes.

```
abeShapesOutside( geGetEditCellView() geGetSelSet() L2 )
```

Related Topics

abeLayerOutside

abeTileIterator

```
abeTileIterator(
     o_abeInputLayer
     [ ?bbox l_bbox ]
     )
     => o_tileIter / nil
```

Description

Returns the tile iterator for the primitive rectangles and polygons on the ABE layer. The iterator can be restricted to a region of the layer.

This means that the function takes an ABE layer and returns an iterator holding a list of tessellated rectangles. If a figure is represented by a polygon in an OpenAccess layer, it is broken into rectangles that do not overlap and have no gaps.

Arguments

o_abeInputLayer	ID of the ABE layer.
?bbox *l_bbox*	Processes only the shapes in the specified region, represented by a list of two window coordinates for the lower-left and the upper-right corners of the region. For example, `list(20:20 220:280)`

Values Returned

o_tileIter	ID of the ABE Layer tile iterator.
`nil`	The function was not successful because ABE is not initialized or the ABE layer does not exist.

Examples

Iterates through the primitive tiles in the `a1` ABE layer and outputs the bounding box for each tile.

```
tileIter = abeTileIterator( a1 )
while(
    tile = tileIter->next
    print( tile->bbox )
)
```

Related Topics

abeIslandIterator

17

Symbolic Placement of Devices Functions

This topic provides a list of Cadence® SKILL functions associated with Symbolic Placement of Devices (SPD).

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- User-Defined Abutment Functions
  - spdCalcOdSpacingWithPoly
  - spdGetAbutName
  - spdGetAbutStrategy
  - spdGetSymDeviceInfo
  - spdGetUserAbutProc
  - spdPerformAbutment
  - spdRegUserAbutProc
  - spdUnregUserAbutProc
- User Flow Callback Functions
  - spdGetUserFlowProc
  - spdRegUserFlowProc
  - spdUnregUserFlowProc

SPD User-Defined Abutment Functions

You can use the user-defined abutment functions to define your own custom abutment callback functions.

For PDK-specific abutments, such as dummy poly abutment, other than the regular oxide diffusion (OD) abutment, you will need to define your own custom abutment callback functions. To enable user-defined abutment, you must load the callback functions required by the spdRegUserAbutProc SKILL function and register them before launching SPD. Otherwise, only oxide diffusion abutment will take place.

Related Topics

Using User-Defined Abutment and Callback Functions in SPD (video)

spdCalcOdSpacingWithPoly

```
spdCalcOdSpacingWithPoly(
     d_lhsInst
     d_rhsInst
     n_minOd
     n_minPoly
     )
     => n_odSpacing
```

Description

Calculates the required oxide diffusion spacing between two adjacent instances based on minimum oxide diffusion spacing and polysilicon spacing constraints.

This function can be used to write a user function that is called when the value of *SPD Options - Generation - X: OD* drop box is set to *UserFunction*. This user callback specifies the required oxide diffusion spacing between two unabutted instances.

Arguments

d_lhsInst	Database ID of the left-hand instance.
d_rhsInst	Database ID of the right-hand instance.
n_minOd	The minimum spacing for the oxide diffusion layer.
n_minPoly	The minimum spacing for the polysilicon layer.

Value Returned

n_odSpacing	The required oxygen diffusion spacing between the left-hand and right-hand instances.

Example

Using user-defined dummy poly spacing callback function:

```
procedure(MyOdSpcProc(lInst rInst)
  let((odSpcWithPoly minOd minPoly)
    minOd = 0.038 ; minimum oxide diffusion spacing constraint
    minPoly = 0.068 ; minimum poly spacing constraint
    odSpcWithPoly = spdCalcOdSpacingWithPoly(lInst rInst minOd minPoly)
```

```
    info("spdCalcOdSpacingWithPoly: lInst %s rInst %s => %f\n" lInst~>name
rInst~>name odSpcWithPoly)
    odSpcWithPoly
  )
)
```

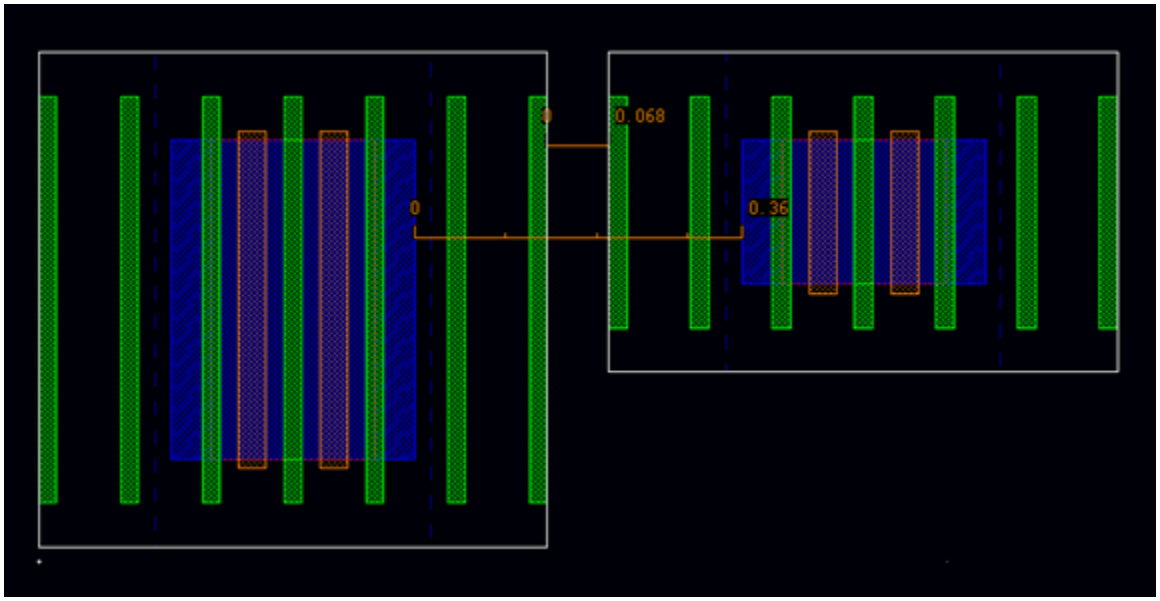
To activate the user callback:

1. In the SPD Editing window, open the *SPD Options* form.
2. On the *Generation* option pane, select *UserFunction* from the *X: OD* drop-down list.
3. Enter `MyOdSpcProc` in the *X: OD* field and click *Apply*.
4. Click *Preview Layout* or *Generate Layout*.

The following message is displayed in CIW:

```
spdCalcOdSpacingWithPoly: lInst I8 rInst I9 => 0.360000
```

The generated layout is as below:



In the figure, the left-hand instance is `I8` and the right-hand instance is `I9`. These are devices with dummy poly.

In this example, the user callback `MyOdSpcProc()` is performed on instances `I8` and `I9` to calculate the minimum required oxide diffusion spacing between them.

Although the minimum oxide diffusion spacing constraint is `0.038`, the oxide diffusion spacing between `I8` and `I9` cannot be `0.038`. This is because additional spacing is required for the minimum poly spacing constraint, which is `0.068`.

You can use function `spdCalcOdSpacingWithPoly()` to calculate the required oxide diffusion spacing that can satisfy both the minimum oxide diffusion and the minimum polysilicon spacing constraints.

In this case, `spdCalcOdSpacingWithPoly()` considers the two instances `I8` and `I9`, minimum oxide diffusion spacing `0.038`, and minimum polysilicon spacing `0.068` as arguments. It returns `0.36` which is the required oxide spacing between `I8` and `I9` to satisfy both spacing constraints. By setting the oxide diffusion spacing of `I8` and `I9` by `0.36`, both the minimum polysilicon spacing constraint `0.068` and the minimum oxide diffusion constraint `0.038` are satisfied.

Related Topics

SPD Generation Options

spdGetAbutName

```
spdGetAbutName(
     x_strategy
     [ ?libName t_libName ]
     )
     => t_abutName / nil
```

Description

Returns the user abutment name associated with the specified abutment strategy registered using the spdGetUserAbutProc function.

Arguments

x_strategy	The abutment strategy. The values `0` and `1` are system-reserved for unabutment and normal oxide diffusion abutment respectively. User abutment strategy numbering starts at `2`.
`?libName` *t_libName*	Name of the library associated with the user abutment. The default value is `nil`, which refers to the global abutment callback.

Value Returned

t_abutName	Name of the user abutment for the specified library associated with the given abutment strategy.
`nil`	No user abutment name was found.

Example

Register the user abutment name and abutment strategy pairs.

```
; abutment name  :  abutment strategy
; "dummyStyle1"  :   2
; "dummyStyle2"  :   3
```

```
spdRegUserAbutProc(list(list("dummyStyle1" 2) list("dummyStyle2" 3))
"MyUserAbutProc" "MyUserAbutCheckProc" "MyUserAbutRestoreProc" ?libName "libA")
=> t
```

Query the user abutment name associated with the abutment strategy `3`and library `libA`.

```
spdGetAbutName(3 ?libName "libA")
=> "dummyStyle2"
```

Related Topics

spdGetUserAbutProc

spdGetAbutStrategy

```
spdGetAbutStrategy(
     t_abutName
     [ ?libName t_libName ]
     )
     => x_strategy / nil
```

Description

Returns the abutment strategy associated with the specified user abutment name registered using the spdRegUserAbutProc function.

Arguments

t_abutName	Name of the user abutment.
?libName *t_libName*	Name of the library associated with the user abutment. The default value is `nil`, which refers to the global abutment callback.

Value Returned

x_strategy	The abutment strategy associated with the specified user abutment name and library.
`nil`	No abutment strategy was found.

Example

Registers the user abutment name and abutment strategy pairs.

```
; abutment name  :  abutment strategy
; "dummyStyle1"  :   2
; "dummyStyle2"  :   3
spdRegUserAbutProc(list(list("dummyStyle1" 2) list("dummyStyle2" 3))
"MyUserAbutProc" "MyUserAbutCheckProc" "MyUserAbutRestoreProc" ?libName "libA")
=> t
```

Query the abutment strategy associated with the user abutment name `dummyStyle2` and library `libA`

```
spdGetAbutStrategy("dummyStyle2" ?libName "libA")
```

```
=> 3
```

Related Topics

spdRegUserAbutProc

Abutment

spdGetSymDeviceInfo

```
spdGetSymDeviceInfo(
     d_inst
     )
     => r_infoObj / nil
```

Description

Returns an object containing information, such as length, width, fingers, and binding related to the specified symbolic device. This information can be used to write user callback functions.

Arguments

d_inst	Database ID of the instance of a symbolic device.

Value Returned

r_infoObj	Information related to the specified symbolic device instance. This information can be queried using the *obj*~>? or *obj*~>?? command. ■ *obj*~>length: Transistor length (*f_value*) ■ *obj*~>width: Transistor width (*f_value*) ■ *obj*~>fingers: Number of fingers (*x_value*) ■ *obj*~>schInst: Source schematic instance to look for additional information. See example for more details about the information returned.
nil	No information related to the specified symbolic device is found.

Example

Return the information related to the symbolic device instance `inst`, where,

- Length is `2e-08`
- Width is `1.2e-07`

- Number of fingers is `1`

The source schematic instance info is `(((db:0x29e5d81a 0 0 0)))`, which is the return value of `bndGetBoundObjects(inst)`. Where, `db:0x29e5d81a` is the database ID of the source schematic instance bound to `inst`.

```
spdGetSymDeviceInfo(inst)~>??
=>
(length 2e-08 width 1.2e-07 finger
    1 schInst
    (((db:0x29e5d81a 0 0 0)))
)
```

Related Topics

bndGetBoundObjects

Device Representations

spdGetUserAbutProc

```
spdGetUserAbutProc(
     ?libName t_libName
     )
     => r_infoObj / nil
```

Description

Returns an object containing the registered user abutment options with associated user callbacks and the description that were registered using the spdRegUserAbutProc function.

Arguments

`?libName` *t_libName*	Name of the library associated with the user abutment. The default value is `nil`, which refers to the global abutment callback.

Value Returned

r_infoObj	The object containing options, associated user abutment callbacks and the description for the specified library. This information can be queried using the `obj~>?` or `obj~>??` command. See example below for details of the information returned.
`nil`	The user abutment callback is not registered.

Example

Register the user abutment and related callbacks.

```
spdRegUserAbutProc(list("dummyStyle") "MyUserAbutProc" "MyUserAbutCheckProc"
"MyUserAbutRestoreProc" ?libName "libA" ?description "user abutment for libA"
?checkOdAbut nil)
=> t
```

Query the information of current registration.

```
spdGetUserAbutProc(?libName "libA")~>??
=>
(abutNames
```

```
    ("dummyStyle") abutProc "MyUserAbutProc" checkProc
    "MyUserAbutCheckProc" restoreProc "MyUserAbutRestoreProc" description "user
abutment for libA" checkOdAbut nil
```

Here,

- The user abutment name is "`dummyStyle`"
- The user callback to perform user abutment in SPD Preview is "`MyUserAbutProc`"
- The user callback to reject an invalid user abutment in SPD Edit is "`MyUserAbutCheckProc`"
- The user callback to restore the user abutment during round-trip from layout to SPD Edit is "`MyUserAbutRestoreProc`"
- The description of the main characteristics for the user callback set is "`user abutment for libA`"
- The value of `checkOdAbut` is `nil`.

Related Topics

`spdRegUserAbutProc`

User-Defined Abutment Callback Functions

spdPerformAbutment

```
spdPerformAbutment(
     d_lhsInst
     d_rhsInst
     x_keepContacts
     t_abutStrategy { sdFirst | dummyFirst }
     )
     => ld_chain / nil
```

Description

Performs abutment on two adjacent instances according to the specified arguments and the SPD P/N chain alignment setting.

Arguments

d_lhsInst	Database ID of the left-hand instance.
d_rhsInst	Database ID of the right-hand instance.
x_keepContact	Controls whether abutment deletes or keeps contacts. Valid Values: `0`: means abutment automatically determines if the contacts should be deleted or kept. `1`: means abutment keeps contacts. `2`: means abutment deletes contacts.
t_abutStrategy	Specifies whether abutment is first performed on the source/drain or the dummy poly. Valid Values: `sdFirst`, abuts first on the source/drain. `dummyFirst`, abuts on the dummy poly. If the dummy poly abutment fails, the software tries to abut on source/drain.

Value Returned

ld_chain	Returns the list of database IDs of abutted instances if the abutment succeeded.
nil	Abutment failed.

Example

Abut `inst1` and `inst2` with the `sdFirst` abutment strategy.

```
spdPerformAbutment(inst1 inst2 0 "sdFirst")
         (db:0x1a594a9a db:0x1a59451a)
```

Related Topics

Abutment

spdRegUserAbutProc

```
spdRegUserAbutProc(
     l_abutNames
     t_abutProc
     t_checkProc
     t_restoreProc
     [ ?libName t_libName ]
     [ ?description t_description ]
     [ ?checkOdAbut t | nil ]
     )
     => t / nil
```

Description

Registers callback to check the design for OD abutment.

Registers a list of user-defined abutments and related callbacks to perform these abutments in the SPD Editing or SPD Preview window or when a layout generated using SPD is brought back to the SPD Editing window. You can register them globally or by name of the library. If the design contains instances from different libraries, SPD loads the user-defined abutments and related callbacks from only one of registered libraries.

Arguments

l_abutNames	A list of user abutment names or name-integer pairs. The integer number represents the abutment strategy associated with the user abutment name. The integer number should start from `2`. `0` and `1` are reserved for unabutment and normal oxide diffusion abutment respectively. If specified, the abutment strategy and the user abutment name can be cross-referenced later by `spdGetAbutStrategy` and `spdGetAbutName` respectively. The user abutment name and abutment strategy are referenced only by the registered callbacks `abutProc`, `checkProc` and `restoreProc` to find out which abutment should be performed. Conceptually, you can assign any value that is suitable for your requirement. However, it is recommended to use the same values defined in the foundry PDK, if available.

t_abutProc

The user callback to perform user abutment in SPD Preview.

```
abutProc(
d_lhsInst
d_rhsInst
r_abutCxt
)
=> l_chain / nil
```

where:

- *d_lhsInst* is database ID of the left-hand instance.
- *d_rhsInst* is database ID of the right-hand instance.

  Note: *d_lhsInst* and *d_rhsInst* are two adjacent real devices.

- *r_abutCxt* is the abutment context, which stores for each SPD window the settings that can be referenced in the user callback.

  Use `obj~>??` to query. For example, `obj~>abutName` returns the name of the user abutment.

Value Returned

- *l_chain*, lists the database IDs of the chained instances.
- `nil`, if abutment was unsuccessful.

t_checkProc

The user callback to reject an invalid user abutment in the SPD Edit window.

```
checkProc(
     d_lhsInst
     d_rhsInst
     r_abutCxt
     )
     => t / nil
```

where:

d_lhsInst is database ID of left-hand instance.

d_rhsInst is database ID of right-hand instance.

d_lhsInst and *d_rhsInst* are two adjacent symbolic devices.

`r_abutCxt` is the abutment context, which stores for each SPD window the settings that can be referenced in the user callback.

Value Returned

- `t`, if the user abutment is valid.
- `nil`, if abutment was unsuccessful..

`t_restoreProc`

The user callback to restore user abutments when a design created using SPD is brought back to the SPD Edit window from layout.

If not specified, only normal oxide diffusion abutment is restored in this process. All other abutments are unabutted in the SPD Editing window.

```
restoreProc(
     d_lhsInst
     d_rhsInst
     [r_abutCxt]
     )
     => t_abutName / t / nil
```

where:

- *`d_lhsInst`* is database ID of left-hand-side instance.
- *`d_rhsInst`* is database ID of right-hand-side instance.

  Note: *`d_lhsInst`* and *`d_rhsInst`* are two adjacent real devices.
- *`r_abutCxt`* is the abutment context, which stores for each SPD window the settings that can be referenced in the user callback.

Value Returned

- *`abutName`* for user abutment.
- `t` for normal oxide diffusion abutment.
- `nil` for unabutment.

[?libName *t_libName]*

Name of the library associated with the user abutment names (*l_abutNames*) and the user abutment callback set (*t_abutProc*, *t_checkProc*, and *t_restoreProc*).

The default value is nil, which means that the user abutment names and the user abutment callback set is global and applies to all libraries that are not registered.

[?description *t_description]*

Describes the main characteristics for the user callback set. This information is useful in deciding whether the callback set upgrade is needed.

?checkOdAbut

When set to t, *t_checkProc* is called during OD abutment to decide whether the OD abutment in SPD Editing window is accepted.

The default value is nil.

Value Returned

t	The user abutments are registered successfully.
nil	The user abutments cannot be registered because the library is already registered. The current registration must be unregistered first.

Examples

Example 1

Check the design for OD abutment.

```
spdRegUserAbutProc(nil nil "myAbutCheckCB" nil)
```

This example works in both modes. When *l_abutNames* is not specified, the function checks for OD abutment. Arguments *t_abutProc* and *t_restorProc* are ignored, even if you specify them in the procedure.

Example 2

For symbolic view, inputs are symbolic devices

```
procedure(MyUserAbutCheckProc(lInst rInst abutCxt)
 let(((delta 1e-9) diff)
   ; Some process node do not allow dummy poly abutment for diffrent gate lengths

   diff = spdGetSymDeviceInfo(lInst)~>length - spdGetSymDeviceInfo(rInst)~>length
   if( abs(diff) >= delta then
        hiDisplayAppDBox(
            ?name               'dBox
            ?dboxBanner         "Warning"
           ?dboxText           sprintf(nil "Failed to abut '%s' and '%s' because
their gate lengths are not equal." lInst~>name rInst~>name)
            ?dialogStyle        'modal
            ?dialogType         hicInformationDialog
            ?buttonLayout       'Close
        )
  nil  ; abutment rejected
 else
  t    ; abutment ok
   )
 ) ; let
)
```

For preview, inputs are real devices

```
procedure(MyUserAbutProc(lInst rInst abutCxt)
  let(((strategy "sdFirst") (abutName abutCxt->abutName))
    ; valid abutStrategies to lxChain api
    ; set env variables for user abutment
    when(abutName == "dummyStyle"
      strategy = "dummyFirst"
    )
    spdPerformAbutment(lInst rInst 0 strategy)
    ; reset env variables for regular oxide diffusion abutment
  ); let
)
```

For round-trip back to SPD from layout, inputs are real devices (lInst = left, rInst = right)

```
procedure(MyUserAbutRestoreProc(lInst rInst)
  let((result)
```

```
    ; Change this param to match real Pcell
    when(atoi(vfoGetParam(lInst "numRightDummies")) >= 1 &&
         atoi(vfoGetParam(rInst "numLeftDummies")) >= 1
       result = "dummyStyle"
    ); when
    result
  ); let
)
; unregister the current user abutment callback before registering a new one.

spdUnregUserAbutProc(?libName "libA")
=> t
```

Register user abutment functions to activate the user abutment.

```
spdRegUserAbutProc(list("dummyStyle") "MyUserAbutProc" "MyUserAbutCheckProc"
"MyUserAbutRestoreProc" ?libName "libA" ?description "description for libA"
?checkOdAbut nil)
=> t
```

Related Topics

spdUnregUserAbutProc

spdGetUserAbutProc

spdGetAbutStrategy

spdGetAbutName

User-Defined Abutment Callback Functions

spdUnregUserAbutProc

```
spdUnregUserAbutProc(
     [ ?libName t_libName ]
     )
     => t / nil
```

Description

Unregisters the current user abutment callback for the specified library. You cannot unregister a callback during an SPD session because it is still in use.

Arguments

`?libName` *`t_libName`*	Name of the library associated with the user abutment to unregister. The default value is `nil`, which refers to the global abutment callback.

Value Returned

`t`	The user abutment callback was unregistered successfully.
`nil`	The user abutment callback cannot be unregistered, because there is no user abutment callback registered for the specified library or the callback is currently being used by an SPD session.

Example

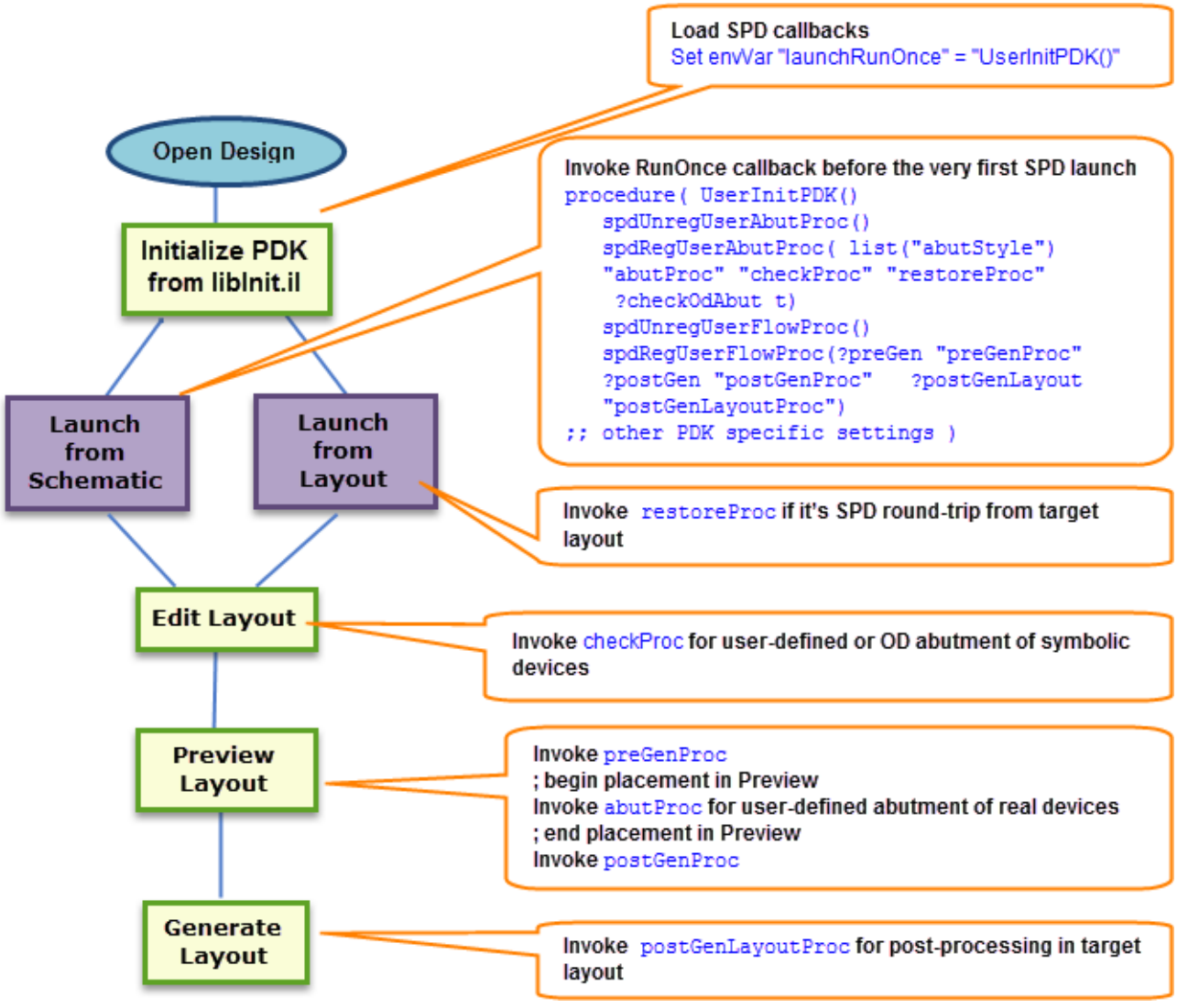
Unregister the current user abutment callback associated with library `libA`.

```
spdUnregUserAbutProc(?libName "libA")
=> t
```

SPD User Flow Callback Functions

Virtuoso Symbolic Placement of Devices (SPD) supports user flow callbacks that help in customizing the design in specific steps of the SPD flow. User flow callback functions help in customizing the design in specific steps of the SPD flow.

The following figure depicts the SPD flow when the user flow callback functions are used.



Related Topics

User-Defined Flow Callback Functions

spdGetUserFlowProc

```
spdGetUserFlowProc(
     )
     => r_infoObj / nil
```

Description

Returns an object containing the user flow callbacks that were registered using the spdRegUserFlowProc function.

Arguments

None

Value Returned

r_infoObj	The object containing user flow callbacks. This information can be queried using the `obj~>?` or `obj~>??` command. See example below for the details of the information returned.
`nil`	The user flow callback is not registered.

Example

Register the user flow callbacks.

```
spdRegUserFlowProc(?postGen "MyPostGenProc" ?preGen "MyPreGenProc")
=> t
```

Query the information of current registration.

```
spdGetUserFlowProc()~>??
=> (postGen "MyPostGenProc" preGen "MyPreGenProc" ?postGenLayout
"MyPostGenLayProc")
```

Here:

- The user callback to post-process real devices in SPD Preview after placement is `MyPostGenProc`.
- The user callback to pre-process real devices in SPD Preview before placement is `MyPreGenProc`.

- The user callback to postprocess the SPD result after the design placed in the target layout is `MyPostGenLayoutProc`.

Related Topics

`spdRegUserFlowProc`

spdRegUserFlowProc

```
spdRegUserFlowProc(
     [ ?postGen t_postGenProc ]
     [ ?preGen t_preGenProc ]
     [ ?postGenLayout t_postGenLayProc ]
     [ ?symPreviewInstParam t_symPreviewInstParamProc ]
     )
     => t / nil
```

Description

Registers user-defined callbacks to further customize the design in specific steps of the SPD flow.

Arguments

?postGen *t_postGenProc*

The user callback to postprocess real devices after placement in the SPD Preview window. For example, enable the dummy poly shapes only on the outer-most source or drain in a row.

```
userPostGenProc(
     d_cellview
     l_instRows
     l_stackedInsts
     [ r_cxtObj ]
     )
     => t / nil
```

where,

- *d_cellview* is the database ID of the cellview displayed in the SPD Preview window.
- *l_instRows* is a list of real devices placed in the rows with the following syntax:

```
l_instRows = list( l_row1 l_row2 ...)
;from bottom to top
l_row = list( l_chain1 l_chain2 … )
;device chains in each row;
l_chain = list( d_inst1 d_inst2 … )
;database IDs of the instances in each device chain;
```

- *l_stackedInsts* is a list of database IDs of the stacked real devices.

- *r_cxtObj* is an object that retrieves additional information related to the design open in SPD through the preview cellview process with the following syntax:

```
cxtObj~>powerRails
cxtObj~>signalTracks
cxtObj~>inst~>rows
cxtObj~>inst~>stacked
cxtObj~>ref~>topOdY, bottomOdY, centerY
     rowX {start, end}
cxtObj~>setting~>trunkWidth, trunkOdSpc,
trunkSpc, rowPairSpc
```

  where,

  - cxtObj~>*powerRails*
    is a list of power and ground shapes.
  - cxtObj~>*signalTracks*
    is list of trunks created by SPD.
    list(*l_channel1 l_channel2*)

    where,
    l_channel
    is
    list(*l_track1 l_track2 l_track3…*)

    where,
    l_track
    is
    list(*l_trunk1 l_trunk2 …*)

- `cxtObj~>inst~>`*`rows`*
  is a list of real devices placed in rows that follow the row pattern from bottom to top.
  `list(` *`l_row1 l_row2 ...`*`)`

  where,
  `l_row`
  is
  `list(` *`l_chain1 l_chain2`* `… ) | nil`
  Device chains in each row or `nil` if the row is empty.

  where,
  `l_chain`
  is
  `list(` *`inst1 inst2`* `… ) |` *`inst`*
  Instances in each device chain or a single instance.

- `cxtObj~>inst~>`*`stacked`*
  is a list of stacked real devices if device stacking is used

- `cxtObj~>ref~>`*`topOdY, bottomOdY, centerY`*
  `rowX{`*`start, end`*`}`
  where,

  - *`topOdY`* is the topmost oxide edge of all instances.
  - *`bottomOdY`* is the lowest oxide edge of all instances.
  - *`centerY`* is the middle of oxide edge of row pairs.
  - *`rowX`* is the leftmost and rightmost oxide edge of each row.

- cxtObj~>setting~>*trunkWidth, trunkOdSpc, trunkSpc, rowPairSpc* where,
  - *trunkWidth* specifies the width of the trunk.
  - *trunkOdSpc* specifies spacing between oxide and trunk.
  - *trunkSpc* specifies spacing between the trunks.
- *rowPairSpc* specifies a list of `within row pair` spacing values and `between row pair` spacing values.

t_preGenProc

The user callback to preprocess real devices in the SPD Preview window before placement. For example, remove unwanted layers from the device.

```
userPreGenProc(
     d_cellview
     )
     => t / nil
```

where,

- *d_cellview* is the database ID of the cellview displayed in the SPD Preview window.

?postGenLayout *t_postGenLayProc*

The user callback to post-process the SPD result after the design is placed in the target layout. For example, MakeCell or add to FigGroup.

Default is `nil`.

```
userPostGenLayoutProc(
     d_cellView
     l_objs
     [ r_genLayCxt ]
     )
     => t / nil
```

Where

- *d_cellview* is the cellview ID of the target layout where SPD generates real devices.
- *l_objs* is a list of all objects generated from SPD. For example, instances, power rails, and signal tracks.
- *r_genLayCxt* is a context object that contains additional genLayout information with the following syntax:

  ```
  cxtPostGenLayoutObj~>otherClone
  ```

  Allows launching SPD from one clone group. This options filters out the following:

  - More than one clone groups.
  - One clone group and other instances. In this case, only the instances are opened in SPD.
  - Other fig groups.

See Example 2 for more information.

`?symPreviewInstParam` *t_symPreviewInstParamProc*

The user callback that specifies device size on the SPD gate size previewer.

```
userSymPreviewInstParamProc(
     d_instance
     )

    => DPL
```

Where

- *d_instance* is the database ID of an SPD instance on the SPD gate size previewer

The callback returns the disembodied property list that contains the following fields to represent device display size on the SPD gate size previewer.

- *f_fingerWidth*
- *f_length*
- *x_fingerCount*

See Example 3 for more information.

Value Returned

`t`	The user flow callback is registered successfully.
`nil`	The user flow callback cannot be registered. The current registration must be unregistered before a new one can be registered.

Examples

Example 1

Specify `MyPostGenProc` cellname as `ABPlusCDBar2`

```
#PowerRails's Net: ("gnd!" "vcc!")
#Ref: (topOdY 1.55 bottomOdY 0.1 centerY (0.75) rowX ((0.07 1.85) (0.07 2.05)))
#Setting: (trunkWidth 0.06 trunkOdSpc 0.06 trunkSpc 0.06 rowPairSpc (1.0 1.0))
#Stacked Instances: nil

#Find 2 tracks from 1 channel
Net:net037 (((0.3275 0.43) (0.4375 0.49)))
Net:net036 (((0.5325 0.43) (0.6425 0.49)))
Net:D (((1.3675 0.43) (1.4775 0.49)))
Net:A (((1.5725 0.43) (1.6825 0.49)))

#Find 1 chain from 2 rows
PM6[MY]--PM4--PM5--PM7[MY]
procedure( MyPostGenProc( d_cellview l_inst_rRows l_stacked_iInsts @optional
cxtObj)
  let( ()
    info("Enter MyPostGenProc: CellName:%s\n" d_cellview~>cellName)
    info("#PowerRails's Net: %L\n",cxtObj~>powerRails~>net~>name)
    info("#Ref: %L\n",cxtObj~>ref~>??)
    info("#Setting: %L\n",cxtObj~>setting~>??)
    info("#Stacked Instances: %L\n",cxtObj~>inst~>stacked~>name)

    ;Print the information at the second track of first channel.
    MyGetTrackData(cxtObj->signalTracks 1 2)

    ;Print the information at the first chain of second row.
    MyGetInstData(cxtObj->inst 2 1)
  )
```

```
);MyPostGenProc

procedure(MyGetTrackData(signalTracks numOfChannel numOfTrack)
  let( (trackList track)
    info("#Find %x track from %x channel\n" numOfTrack numOfChannel)
    ;Get trackList from specified channel
    trackList = nth(numOfChannel-1 signalTracks)
    ;Get track from trackList
    track = nth(numOfTrack-1 trackList)

    ;Print net name
    foreach(trunk track
      info("Net:%s (%L)\n" trunk~>net~>name trunk~>bBox)
    )

  );let
);MyGetTrackData

procedure(MyGetInstData(instObj numOfRow numOfChain )
  let( (row chain pre)
    info("#Find the %x chain from %x row\n" numOfChain numOfRow);
    ;Get the specified row
    row = nth(numOfRow-1 instObj->rows)
    ;Get the specified chain
    chain = nth(numOfChain-1 row)

    foreach(inst chain
      when(pre info("--"))
      info("%s" inst->name)
      when(inst~>orient != "R0"
        info("[%s]" inst~>orient)
      )
      pre = inst
    );inst

  ) ;let
); MyGetInstData
```

Example 2

The following procedure is default callback for the clone group:

```
procedure( MyPostGenLayProc( cv objs @optional cxtObj )
  let( (figGroup figGroupList)
    info("Enter MyPostGenLayoutProc\n")
    when( length(cxtObj~>otherClone) > 0
      info("%d otherClones detected.\n",length(cxtObj~>otherClone))
      ;Delete Original Clone Family
      lxDeleteSynchronousCloneFamily(cxtObj~>otherClone)
      geDeselectAll(geGetCellViewWindow(cv))

      ;Create New Fig Group
      geSelectFigs(objs)
      leHiCreateGroup()
      figGroup = car(geGetSelectedSet())
      geDeselectAll(geGetCellViewWindow(cv))
      ;info("fig Group:%L, \n", figGroup)
    ) ; when
  );let
);procudure
spdRegUserFlowProc(?postGenLayout "MyPostGenLayProc" )
```

Example 3

Specifies device size on each instance for the SPD gate size previewer.

```
procedure(MySymPreviewInstParam( inst )
    info("MySymPreviewInstParam(%s)" inst->name)
    let((ret)
        ret = ncons(nil)
        ret->fingerWidth    = 0.1 /* float */
        ret->length         = 0.02 /* float */
        ret->fingerCount    = 2 /*int*/
        info("%L\n" ret)
        ret
        )
); => DPL
spdRegUserFlowProc(?symPreviewInstParam "MySymPreviewInstParam")
```

spdUnregUserFlowProc

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

Description

Unregisters the current user flow callback. You cannot unregister a callback during an SPD session because it is still in use.

Arguments

None

Value Returned

`t`	The user flow callback was unregistered successfully.
`nil`	The user flow callback cannot be unregistered because there is no user flow callback registered, or the callback is currently being used by an SPD session.

Example

The current user flow callback is unregistered.

```
spdUnregUserFlowProc()
=> t
```

18

Virtuoso Placer Functions (Virtuoso Layout Suite EXL)

This topic provides a list of Cadence® SKILL functions associated with Virtuoso Placer.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

- lobGetRegisteredFillOverrideParameters
- lobGetRegUserProc
- lobRegisterFillOverrideParameters
- lobRegUserProc
- lobUnregisterFillOverrideParameters
- lobUnRegUserProc

lobGetRegisteredFillOverrideParameters

```
lobGetRegisteredFillOverrideParameters(
     [ t_libName ]
     [ t_cellName ]
     )
     => l_paramNameValue / nil
```

Description

(Virtuoso Layout Suite EXL) Returns the first set of registered fill override parameters that matches the specified library and cell names.

Arguments

t_libName	Specifies the name of the library that contains the fill overrides. If not specified or set to "", all currently registered fill overrides are listed.
t_cellName	Specifies the cell name to check for registered fill overrides. If not specified, all cell overrides that are registered for the given library are listed.

Value Returned

l_paramNameValue

If matching cell and library names are found, a list of `paramName paramValue` pairs is returned.

If both library and cell names are non empty, only the parameters of the first registered match are returned.

```
list(list(paramName1 paramValue1) ... list(paramNameN
paramValueN))
```

If the cell name is empty, all currently registered cellRegex values and their parameters are returned.

```
list(
list( cellname1              list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cellName2              list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cell*2                 list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cell[nN]ame3           list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
)
```

If both library and cell names are "", all currently registered library and cellRegex parameters are returned.

```
list(
list(libName1
list(
list( cellname1             list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cellName2             list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cell*2                list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cell[nN]ame3          list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
)
)
list(libName2
list(
list( cellname1             list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cellName2             list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cell*2                list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
list( cell[nN]ame3          list(list(paramName1
paramValue1) ... list(paramNameN paramValueN)))
)
)
)
```

`nil` The command was unsuccessful.

Examples

The following examples register fill overrides for three libraries and then retrieve the registered fill override parameters.

```
lobRegisterFillOverrideParameters("libName1" "cellname1" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE")))
=> t
lobRegisterFillOverrideParameters("libName1" "cellName2" list(list(
"boolStrParam1" "FALSE") list("boolStrParam2" "FALSE")))
=> t
lobRegisterFillOverrideParameters("libName2" "cell*2" list(list( "boolStrParam3"
"FALSE") list("boolStrParam4" "FALSE")))
=> t
```

```
lobRegisterFillOverrideParameters("libName3" "cell[nN]ame3" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
=> t


lobGetRegisteredFillOverrideParameters("libName1")
=> list(list(cellname1 list(list(list("boolStrParam1" "TRUE") list("boolStrParam2"
"TRUE")))
list(list( cellname2 list(list(list("boolStrParam1" "FALSE") list("boolStrParam2"
"FALSE")))


lobGetRegisteredFillOverrideParameters("libName2" "cellAnyStringHere2")
=> list(list("cell*2" list(list( "boolStrParam3" "FALSE") list("boolStrParam4"
"FALSE")))


lobGetRegisteredFillOverrideParameters()
=> list(list("cell[nN]ame3" list(list( "boolStrParam3" "FALSE")
list("boolStrParam4" "FALSE")))
```

Related Topics

lobRegisterFillOverrideParameters

lobUnregisterFillOverrideParameters

lobGetRegUserProc

```
lobGetRegUserProc(
     g_userProc
     )
     => l_userProcsString / nil
```

Description

(Virtuoso Layout Suite EXL) Returns a list of strings that contain the name of the specified registered user procedure.

Arguments

g_userProc	Specifies the name of the registered user procedure to filter strings. Valid symbols are: ‘lobGetAdjacentFillDefsProc, ‘lobGetTransitionFillDefsProc, and ‘lobDummyNetNameProc.

Value Returned

l_userProcsString	A string or a list of strings containing the name of the registered user procedure.
nil	The command was unsuccessful.

Example

In the following example, cdn20FFAdjacentFillDefs stores a user function symbol. This is passed as an argument in lobRegisterUserProc to register the user procedure. Then, lobGetRegUserProc is run to determine the name of the function that is registered:

```
procedure( cdn20FFAdjacentFillDefs()
    ptapfill = '(nil
        name "ptapfill"
        diffusionLPP ("Active" "drawing")
        fingerLPPs ( ("Poly" "drawing") )
        libName "cdn20FF"
        cellName "ptap"
        viewName "layout"
        type      "instance"
```

```
    )

    ntapfill = '(nil
        name "ntapfill"
        diffusionLPP ("Active" "drawing")
        fingerLPPs ( ("Poly" "drawing") )
        libName "cdn20FF"
        cellName "ntap"
        viewName "layout"
        type      "instance"
    )
    i = 2
    list(ntapfill ptapfill)
)

lobRegUserProc(@lobGetAdjacentFillDefsProc 'cdn20FFAdjacentFillDefs)
lobGetRegUserProc('lobGetAdjacentFillDefsProc)
"cdn20FFAdjacentFillDefs"
```

Related Topics

lobRegUserProc

lobUnRegUserProc

lobRegisterFillOverrideParameters

```
lobRegisterFillOverrideParameters(
     t_libName
     t_cellNameOrRegex
     l_paramsList
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Registers parameter overrides to be used by dummy fill. Lookup processes parameter overrides in the order in which they are registered.

Arguments

t_libName	Specifies the name of the library that contains the instance cellviews to override.
t_cellNameOrRegex	Specifies the string against which the cell names are to be matched. If an exact match is not found, the argument follows the SKILL rexMatchp style.
l_paramsList	Name and value pairs of parameters to be overriden.

Value Returned

t	The parameter overrides were registered.
nil	The command was unsuccessful.

Examples

This following examples register overrides for three libraries.

```
lobRegisterFillOverrideParameters("libName1" "NcellName" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE")))
=> t
lobRegisterFillOverrideParameters("libName1" "PcellName" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE") list("intParam" "3")))
=> t
lobRegisterFillOverrideParameters("libName1" "PcellName" list(list(
"boolStrParam1" "TRUE") list("boolStrParam2" "TRUE") list("intParam" "5")))
=> t
```

```
lobRegisterFillOverrideParameters("libName1" "[A-Z]cellName" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
=> t
lobRegisterFillOverrideParameters("libName2" "NcellName" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
=> t
lobRegisterFillOverrideParameters("libName3" "[a-z]*cellName" list(list(
"boolStrParam3" "FALSE") list("boolStrParam4" "FALSE")))
=> t
```

The following example checks for the registered overrides.

```
lobGetRegisteredFillOverrideParameters("libName1" "PcellName")
=> list(list( "boolStrParam1" "TRUE") list("boolStrParam2" "TRUE") list("intParam"
"3"))
```

Related Topics

lobGetRegisteredFillOverrideParameters

lobUnregisterFillOverrideParameters

lobRegUserProc

```
lobRegUserProc(
     g_userFunction
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Registers a user-defined function for a specific purpose, as needed by layout objects. This function is currently used to register procedures to retrieve adjacent fill definitions.

Arguments

g_userFunction

Specifies the symbol of the function to be registered. Valid values are: `@lobGetAdjacentFillDefsProc`, `@lobGetTransitionFillDefsProc`, and `@lobDummyNetNameProc`.

Value Returned

`t`	The key parameters are passed and the function is registered.
`nil`	The key parameters were not passed.

Example

In the following example, a user-defined function `cdn10GetAdjacentFillDefs` is registered.

```
lobRegUserProc(?lobGetAdjacentFillDefsProc 'cdn10GetAdjacentFillDefs)
```

lobUnregisterFillOverrideParameters

```
lobUnregisterFillOverrideParameters(
     t_libName
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Unregisters either all registered fill parameter overrides for the given library name or all the currently registered fill parameters for all libraries.

Arguments

t_libName	Specifies the name of the library that contains the fill parameter overrides to be unregistered.

Value Returned

t	The fill parameter overrides were unregistered.
nil	The command was unsuccessful.

Example

The following example registers overrides for three different libraries and then unregisters overrides for one of the libraries. After checking for registered overrides, all overrides are unregistered followed by a check for all registered overrides.

```
lobRegisterFillOverrideParameters("libName1" "nCellName" list(list(
"boolParamStr1" "TRUE") list("boolParamStr2" "TRUE")))
lobRegisterFillOverrideParameters("libName1" "[a-z]*CellName" list(list(
"boolParamStr1" "FALSE") list("boolParamStr2" "FALSE")))
lobRegisterFillOverrideParameters("libName2" "pCellName" list(list(
"boolParamStr3" "FALSE") list("boolParamStr4" "FALSE")))
lobRegisterFillOverrideParameters("testLibName" "[a-z]*CellName" list(list(
"boolParamStr3" "FALSE") list("boolParamStr4" "FALSE")))


lobUnregisterFillOverrideParameters("libName1")
lobGetRegisteredFillOverrideParameters()


lobUnregisterFillOverrideParameters()
lobGetRegisteredFillOverrideParameters()
```

Related Topics

lobGetRegisteredFillOverrideParameters

lobRegisterFillOverrideParameters

lobUnRegUserProc

```
lobUnRegUserProc(
     g_userFunction
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Accepts user functions based on keyword and unregisters them.

Arguments

g_userFunction	Specifies the user functions to be unregistered. Valid values are: `@lobGetAdjacentFillDefsProc`, `@lobGetTransitionFillDefsProc`, and `@lobDummyNetNameProc`.

Value Returned

`t`	The specified user function was unregistered.
`nil`	The command was unsuccessful.

Example

In the following example, `cdn20FFAdjacentFillDefs` stores a user function symbol. This is passed as an argument in `lobRegisterUserProc` to register the procedure. Then, `lobUnRegUserProc` is run to unregister the function:

```
procedure( cdn20FFAdjacentFillDefs()
    ptapfill = '(nil
        name "ptapfill"
        diffusionLPP ("Active" "drawing")
        fingerLPPs ( ("Poly" "drawing") )
        libName "cdn20FF"
        cellName "ptap"
        viewName "layout"
        type      "instance"
    )
```

```
    ntapfill = '(nil
        name "ntapfill"
        diffusionLPP ("Active" "drawing")
        fingerLPPs ( ("Poly" "drawing") )
        libName "cdn20FF"
        cellName "ntap"
        viewName "layout"
        type      "instance"
    )
    i = 2
    list(ntapfill ptapfill)
)
lobRegUserProc(@lobGetAdjacentFillDefsProc 'cdn20FFAdjacentFillDefs)
lobUnRegUserProc(@lobGetAdjacentFillDefsProc 'cdn20FFAdjacentFillDefs)
```

Related Topics

lobGetRegUserProc

lobRegUserProc

19

Virtuoso Automated Placement and Routing SKILL Functions

This section lists the Cadence® SKILL functions associated with the Virtuoso® Automated Placement and Routing Technologies.

Only the functions documented in this chapter are supported for public use. Any other function, regardless of its name or prefix, and undocumented aspects of the functions described below, are private and subject to change at any time.

The SKILL functions that let you manage tasks in the Virtuoso Automated Placement and Routing Technologies flow can be broadly classified into the following categories:

Virtuoso Device Placement SKILL Functions

- apAdjustBoundary
- apCapturePlacement
- apCreateDeviceRowRegion
- apCustomPDKSettings
- apDeleteDeviceRowRegions
- apDeleteFill
- apDeleteTrims
- apEnableM0OnFlow
- apFixColors
- apHighlightDevices
- apInsertFill
- apInsertTrims
- apIPRegisterCustomMenuItem

- apIPUnregisterCustomMenuItem
- apIPUnregisterCustomMenuItems
- apLoadOptions
- apPDKSetupGetCdsEnvs
- apPDKSetupGetFillParams
- apPDKSetupGetModgenDummyAlias
- apPDKSetupGetModgenDummyParams
- apPDKSetupGetParamFunc
- apPDKSetupGetParamList
- apPDKSetupGetRegProcs
- apPDKSetupGetVtNameList
- apPlaceAuto
- apPlaceAutoMatureNode
- apResetPDKSetup
- apRunBackannotate
- apRunCreateGR
- apRunDeleteGR
- apRunDeviceGeneration
- apSnapInsts
- apSwitchPlacerType
- apUnhighlightDevices
- apValidateTrims
- lobAddCopyFill
- lobIsCopyFill
- lobRemoveCopyFill

Virtuoso Standard Cell Placement SKILL Functions

- vreCapturePlacement
- vreCreateRowRegion
- vreDeletePhysicalCells
- vreDeleteRowRegions
- vreRunGeneration
- vreRunPlacer
- vreShowPlacerLog
- vreStopPlacer
- vreStopRowRegionCreation

Virtuoso Routing Environment SKILL Functions

- vreDeletePowerRouting
- vreDeleteSignalRouting
- vreFinishRouting
- vreGenerateWSPs
- vreGetCellView
- vreGetHandle
- vreGetOption
- vreGetOptions
- vreGetRouter
- vreGetRoutingStyle
- vreIsOption
- vreLoadPreset
- vreRaiseConstraintManager
- vreRaisePreRoutingBrowser
- vreRaiseResultsBrowser
- vreRemoveJogs

- vreRemoveNotches
- vreRunAssistedRouter
- vreRunChecker
- vreRunMeshRouter
- vreRunP2TRouter
- vreRunPowerRouter
- vreRunSignalRouter
- vreSetOption
- vreSetOptions
- vreSetRouter
- vreSetRoutingStatus
- vreSetRoutingStyle
- vreShowCheckerLog
- vreShowSignalRouterLog
- vreStopSignalRouter
- vreStopWSPGeneration

Virtuoso Routing Technology SKILL Functions

- vrtCheckDesign
- vrtCreateIDLayer
- vrtDeleteNets
- vrtPowerRoute
- vrtRouteAssisted
- vrtRouteDesign
- vrtStrandRoute

apAdjustBoundary

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

Description

(Virtuoso Layout Suite EXL and above) Resizes the PR boundary such that it covers any devices that were earlier placed outside the PR boundary. Run this command after running the placer to adjust the PR boundary.

Arguments

None

Value Returned

`t`	The PR boundary was adjusted.
`nil`	The command was unsuccessful.

Examples

Adjusts the PR boundary after running the placer.

```
apAdjustBoundary()
```

Related Topics

apCapturePlacement

Auto P&R Assistant User Interface for Device-Level Placement

apCapturePlacement

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

Description

(Virtuoso Layout Suite MXL) Captures placement data from the current (source) layout and stores it in a migration directory.

Arguments

None

Value Returned

`t`	Placement data is captured from the source layout.
`nil`	The command was unsuccessful.

Examples

Captures placement data from the current layout.

```
apCapturePlacement()
```

Related Topics

apAdjustBoundary

Auto P&R Assistant User Interface for Device-Level Placement

apCreateDeviceRowRegion

```
apCreateDeviceRowRegion(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => d_rowRegionId / nil
```

Description

(Virtuoso Layout Suite EXL and above) Creates a new row region for the specified placement style or engine. The row region is created based on the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview for row region creation.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

d_rowRegionId	Database ID of the row region created.
`nil`	No row region was created.

Examples

Uses the options specified in the Auto P&R assistant for device style placement.

```
cv = geGetEditCellView()
apCreateDeviceRowRegion(cv 'device 'ap)
```

Uses the options specified in batch mode for device style placement.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'device 'ap)
vreSetOption(handle 'setup_createRowRegion t)
apCreateDeviceRowRegion(cv 'device 'ap)
```

Related Topics

apDeleteDeviceRowRegions

Auto P&R Assistant User Interface for Device-Level Placement

apCustomPDKSettings

```
apCustomPDKSettings(
     t_pdkName
     [ ?paramList l_paramsList | ?paramFunc s_paramFunc ]
     [ ?cdsEnvs l_cdsEnvars ]
     [ ?regFile t_regFile ]
     [ ?fillParams l_fillParams ]
     [ ?modgenDummyParams l_ModgenDummyParams ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Registers PDK-specific instance parameters, environment variables, custom constraint registration procedures, fill instance parameters, and Modgen dummy parameters that must be honored while running each step of the automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which parameters are to be registered.

?paramList *l_paramsList*

A list of master cellview names and their CDF parameters. The parameters are set on instances of the master in the given design after generating layout instances from the schematic.

Either specify ?paramList or ?paramFunc. If both are specified, ?paramFunc is honored.

?paramFunc *s_paramFunc*

A symbol of a function that returns a list of CDF parameter key-value pairs to be set. The function argument must be a layout instance that can be used to determine the parameter settings to return.

Either specify ?paramList or ?paramFunc.

?paramList allows specification of CDF parameters based on the instance cell name, whereas ?paramFunc allows specification of CDF parameters based on any instance data.

If both arguments are specified, ?paramFunc is honored.

`?cdsEnvs` *`l_cdsEnvars`*

A list of environment variables to be set at the start of the automated placement and routing flow.

`?regFile` *`t_regFile`*

Path to the text file that contains the procedure to register different device types.

`?fillParams` *`l_fillParams`*

A list of master names and their fill parameters to be set during the dummy fill generation step.

`?modgenDummyParams` *`l_ModgenDummyParams`*

A list of Modgen dummy names and their parameters to be set during the dummy generation step.

Value Returned

`t`	The PDK-specific parameters were registered.
`nil`	The command was unsuccessful.

Examples

Registers instance parameters, environment variables, custom constraint registration procedures, and fill instances parameters for a customized PDK. These parameters are honored in different steps of the automated placement and routing flow.

```
apCustomPDKSettings("customPDK"
                    ?paramList
                        list(
                            list("cell_name1"
                                list(list("param1" "OFF")
                                    list("param2" "ON")
                                )
                            )
                            list("cell_name2"
                                list(list("param2" "OFF")
                                    list("param3" "OFF")
                                )
                            )
                            )
                    ?cdsEnvs
                        list(
                            list(list("tool1[.partition1]") list("varName1")
                                list('type) list(value1)
                            )
```

```
            list(list("tool2[.partition2]") list("varName2")
                list('type2) list(value2)
            )
        )

    ?regFile
        "filepath"
        ?fillParams
        list(
            list("cell_name1"
                list(list("param1" "OFF")
                    list("param4" "ON")
                )
            )
            list("cell_name3"
                list(list("param1" "OFF")
                    list("param2" "OFF")
                )
            )
        )
        ?modgenDummyParams
        list(
            list("cell_name1"
                list(list("dummy1" "OFF")
                    list("dummy4" "ON")
                )
            )
            list("cell_name3"
                list(list("dummy1" "OFF")
                    list("dummy2" "OFF")
                )
            )
        )
```

Related Topics

apPDKSetupGetCdsEnvs

apPDKSetupGetFillParams

apPDKSetupGetModgenDummyParams

apPDKSetupGetParamFunc

apPDKSetupGetParamList

apPDKSetupGetRegProcs

apPDKSetupGetVtNameList

apDeleteDeviceRowRegions

```
apDeleteDeviceRowRegions(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Deletes all automatically created rows and row regions for the specified placement style or engine. The function honors the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview for row region deletion.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	The row regions were deleted.
`nil`	Row regions were not deleted.

Examples

Uses the options in the Auto P&R Assistant for device style.

```
cv = geGetEditCellView()
apDeleteDeviceRowRegions(cv 'device 'ap)
```

Uses the options in batch mode for device style.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'device 'ap)
vreSetOption(handle 'setup_createRowRegion t)
apDeleteDeviceRowRegions(cv 'device 'ap)
```

Related Topics

apCreateDeviceRowRegion

Auto P&R Assistant User Interface for Device-Level Placement

apDeleteFill

```
apDeleteFill(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Deletes all fill from the specified layout for the specified placement style or engine. The function honors the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview for fill deletion.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	Fill were deleted.
`nil`	Fill were not deleted.

Examples

Uses the options in the Auto P&R Assistant for device style for deleting fill from the given cellview.

```
cv = geGetEditCellView()
     apDeleteFill(cv 'device 'ap)
```

Related Topics

apInsertFill

Auto P&R Assistant User Interface for Device-Level Placement

apDeleteTrims

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

Description

(Virtuoso Layout Suite EXL and above) Deletes trims from the current cellview.

Arguments

None

Value Returned

`t`	Trims were deleted.
`nil`	Trims were not deleted.

Examples

Deletes trims from the current cellview.

```
apDeleteTrims()
```

Related Topics

apInsertTrims

Auto P&R Assistant User Interface for Device-Level Placement

apEnableM0OnFlow

```
apEnableM0OnFlow(
     g_enableStatus
     [ w_windowID ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Forces M0 flow to be turned on or off. Typically, the tool checks the Pcell parameters to automatically determine whether the flow must be turned on or off. You can use this SKILL API to force start or stop the flow. This API is valid only in certain advanced node flows.

Arguments

g_enableStatus

Status of M0 flow. The default value depends on the Pcell parameters in the current design.

w_windowID

Window ID of the applicable window. If not specified, the active window is considered.

Value Returned

`t`	M0 flow is turned on or off, as specified.
`nil`	The command was unsuccessful.

Examples

Turns on M0 flow on the current window.

```
apEnableM0OnFlow('t)
```

apFixColors

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

Description

(Virtuoso Layout Suite EXL and above) Swaps colors of flipped rows.

Arguments

None

Value Returned

`t`	Colors are swapped.
`nil`	Colors could not be swapped.

Examples

Swaps colors of flipped rows in M0 On Flow.

```
apFixColors()
```

Related Topics

apEnableM0OnFlow

apHighlightDevices

```
apHighlightDevices(
     d_cellViewID
     [ ?dummy g_dummies ]
     [ ?active g_activeDevices ]
     )
     => t / nil
```

Description

Highlights the selected types of devices in the layout. You can highlight dummy devices, active devices, or both.

Arguments

d_cellViewID

Database ID for the cellview in which devices are to be highlighted.

?dummy *g_dummies*

Specifies whether dummy devices are to be highlighted in the layout.

?active *g_activeDevices*

Specifies whether active devices are to be highlighted in the layout.

Value Returned

t	The specified devices were highlighted.
nil	The command was unsuccessful.

Examples

Highlights all dummy devices in the current cellview.

```
cv = geGetEditCellView()
apHighlightDevices(cv ?dummy t ?active nil)
```

Related Topics

apUnhighlightDevices

apInsertFill

```
apInsertFill
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Inserts fill in the given layout based on the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview for fill insertion.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	Fill was inserted in the given layout.
`nil`	The command was unsuccessful.

Examples

Uses the options specified in the Auto P&R assistant for device style placement.

```
cv = geGetEditCellView()

apInsertFill(cv 'device 'ap)
```

Related Topics

apDeleteFill

Auto P&R Assistant User Interface for Device-Level Placement

apInsertTrims

```
apInsertTrims
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Inserts trims in the current cellview to fix shorts.

Arguments

None

Value Returned

`t`	Trims were inserted in the given cellview.
`nil`	The command was unsuccessful.

Examples

Inserts trims in the current cellview.

```
apInsertTrims()
```

Related Topics

apDeleteTrims

Auto P&R Assistant User Interface for Device-Level Placement

apIPRegisterCustomMenuItem

```
apIPRegisterCustomMenuItem(
     t_commandName
     t_functionName
     [ ?inst g_inst ]
     [ ?modgen g_modgen ]
     [ ?figGroup g_figGroup ]
     )
     => t / nil
```

Description

Registers a custom context menu item in the interactive placer context menu.

Arguments

t_commandName	Specifies the new context menu name.
t_functionName	Specifies the name of the function to be called.
[?inst *g_inst*]	Displays the custom context menu only when the select set contains an instance.
[?modgen *g_modgen*]	Displays the custom context menu only when the select set contains a Modgen.
[?figGroup *g_figGroup*]	Displays the custom context menu only when the select set contains a figGroup.

Value Returned

`t`	The custom context menu item is registered in the interactive placer context menu.
`nil`	The custom context menu could not be registered.

Examples

If the selected set contains instances or Modgens, the context menu includes the custom button `ButtonText` that calls `buttonFunction` on `click`.

```
apIPRegisterCustomMenuItem("ButtonText" "buttonFunction()" ?inst t ?modgen t)
```

Related Topics

apIPUnregisterCustomMenuItem

apIPUnregisterCustomMenuItems

Auto P&R Assistant User Interface for Device-Level Placement

apIPUnregisterCustomMenuItem

```
apIPUnregisterCustomMenuItem(
     t_commandName
     )
     => t / nil
```

Description

Removes the specified custom menu item from the interactive placer context menu.

Arguments

t_commandName	Specifies the name of the custom context menu item to be removed.

Value Returned

t	The specified custom context menu item was removed from the context menu.
nil	The specified custom context menu item does not exist.

Examples

Removes the custom button ButtonText from the context menu.

```
apIPUnregisterCustomMenuItem("ButtonText")
```

Related Topics

apIPRegisterCustomMenuItem

apIPUnregisterCustomMenuItems

Auto P&R Assistant User Interface for Device-Level Placement

apIPUnregisterCustomMenuItems

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

Description

Removes all custom menu items from the interactive placer context menu.

Arguments

None

Value Returned

`t`	All custom context menu items were removed from the interactive placer context menu.
`nil`	The command was unsuccessful.

Examples

Removes all custom context menu items from the interactive placer context menu.

```
apIPUnregisterCustomMenuItems()
```

Related Topics

apIPRegisterCustomMenuItem

apIPUnregisterCustomMenuItem

Auto P&R Assistant User Interface for Device-Level Placement

apLoadOptions

```
apLoadOptions(
     ?ecv d_cv
     ?optFile t_optionsFileName
     ?placerType S_placementStyle
     )
     => t / nil
```

Description

Load options from the given options file into the Auto P&R assistant for the specified placement style.

Arguments

`?ecv` *`d_cv`*

The layout cellview for which the settings are applicable.

`?optFile` *`t_optionsFileName`*

Path and name of the options file.

`?placerType` *`S_placementStyle`*

Placement type for which the settings are to be applied. The valid values are `"device"` and `"stdCell"`.

Value Returned

`t`	The options from the file were loaded to the UI.
`nil`	The options could not be loaded.

Examples

Loads settings from the file `.cadence/dfII/APR/1/presets/initTab.stdCell.gp` for standard cell flow in current cellview.

```
apLoadOptions( ?ecv geGetEditCellView() ?optFile ".cadence/dfII/APR/1/presets/
initTab.stdCell.gp" ?placerType "stdCell")
=> t
```

apPDKSetupGetCdsEnvs

```
apPDKSetupGetCdsEnvs(
     t_pdkName
     )
     => l_envarInfo / nil
```

Description

(Virtuoso Layout Suite EXL) Retrieves a list of CDS environment variables with values that are registered to the specified PDK for the automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which environment variables with values must be retrieved.

Value Returned

l_envarInfo	List of CDS environment variables with values registered for the specified PDK.
nil	There are no registered CDS environment variables for the specified PDK.

Examples

Applies custom PDK settings and then retrieves a list of CDS environment variables with values for the specified PDK.

```
apCustomPDKSettings("CustomPDK" ?cdsEnvs list(list(list("tool1[.partition1]")
list("varName1") list('type) list(value1))))
=> t
apPDKSetupGetCdsEnvs("CustomPDK")
=>((("tool1[.partition1]")
               ("varName1")
               type()
               (value1)
    )
)
```

apPDKSetupGetFillParams

```
apPDKSetupGetFillParams(
     t_pdkName
     )
     => l_paramList / nil
```

Description

(Virtuoso Layout Suite EXL) Retrieves a list of instance fill parameters that are registered to the specified PDK for the automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which fill parameters must be retrieved.

Value Returned

l_paramList	A list of fill parameters registered for the specified PDK.
nil	There are no registered fill parameters for the specified PDK.

Examples

Applies custom PDK settings and retrieves a list of fill parameters for the specified PDK.

```
apCustomPDKSettings("CustomPDK" ?FillParams list(list("cell_name1")
list(list("param1” "OFF")))))
=> t
apPDKSetupGetFillParams("CustomPDK")
=> (("cell_name1"
               (("param1") (“OFF”))
    )
)
```

apPDKSetupGetModgenDummyAlias

```
apPDKSetupGetModgenDummyAlias(
     t_pdkName
     )
     => l_paramList / nil
```

Description

(Virtuoso Layout Suite EXL and above) Retrieves a list of PDK-specific Modgen dummy library, cell, and parameter name aliases registered for the specified PDK in the Virtuoso automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which Modgen dummy library, cell, and parameter name aliases must be retrieved.

Value Returned

l_paramList	A list of lists of Modgen dummy library, cell, and parameter names.
nil	There are no registered dummy library, cell, and parameter names for the specified PDK.

Examples

Retrieves a list of PDK-specific Modgen dummy library, cell, and parameter name aliases registered for customPDK.

```
apPDKSetupGetModgenDummyAlias("customPDK")
```

apPDKSetupGetModgenDummyParams

```
apPDKSetupGetModgenDummyParams(
     t_pdkName
     )
     => l_paramList / nil
```

Description

(Virtuoso Layout Suite EXL and above) Retrieves a list of Modgen dummy instance parameters that are registered to the specified PDK for the automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which Modgen dummy instance parameters must be retrieved.

Value Returned

l_paramList	A list of Modgen dummy instance parameters registered for the specified PDK.
nil	There are no registered Modgen dummy instance parameters for the specified PDK.

Examples

Applies custom PDK settings and retrieves a list of Modgen dummy instance parameters for the specified PDK.

```
apCustomPDKSettings("CustomPDK" ?modgenDummyParams list(list("cell_name1")
list(list("ModgenDummy1" "OFF")))))
=> t
apPDKSetupGetModgenDummyParams("CustomPDK")
=> (("cell_name1"
               (("ModgenDummy1") ("OFF"))
    )
)
```

apPDKSetupGetParamFunc

```
apPDKSetupGetParamFunc(
     t_pdkName
     )
     => s_paramFunc / nil
```

Description

(Virtuoso Layout Suite EXL) Retrieves the PDK-specific instance parameter function that is registered to the specified PDK for the automated placement and routing flow.

Arguments

t_pdkName	Name of the PDK for which instance parameter function must be retrieved.

Value Returned

s_paramFunc	Symbol of the function that determines the CDF instance parameters to apply to the instance after generating layout instances from schematic.
nil	There is no parameter-determining function registered for the specified PDK.

Examples

Retrieves the symbol of the parameter-determining function registered for the specified PDK.

```
apPDKSetupGetParamFunc("CustomPDK")
```

Related Topics

apCustomPDKSettings

apPDKSetupGetParamList

apPDKSetupGetRegProcs

apPDKSetupGetParamList

```
apPDKSetupGetParamList(
     t_pdkName
     )
     => l_paramList / nil
```

Description

(Virtuoso Layout Suite EXL) Retrieves a list of instance parameters that are registered to the specified PDK for the automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which instance parameters must be retrieved.

Value Returned

l_paramList	A list of instance parameters registered for the specified PDK.
nil	There are no registered instance parameters for the specified PDK.

Examples

Applies custom PDK settings and retrieves a list of instance parameters for the specified PDK.

```
apCustomPDKSettings("CustomPDK" ?paramList list(list("cell_name1")
list(list("param1" "OFF"))))
=> t
apPDKSetupGetParamList("CustomPDK")
=> (("cell_name1"
               (("param1") ("OFF"))
   )
)
```

apPDKSetupGetRegProcs

```
apPDKSetupGetRegProcs(
     t_pdkName
     )
     => l_procList / nil
```

Description

(Virtuoso Layout Suite EXL) Retrieves a list of custom constraint registration procedures that are registered to the specified PDK for the automated placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which registered custom constraint registration procedures must be retrieved.

Value Returned

l_procList	A list of custom constraint registration procedures registered for the specified PDK.
nil	There are no registered custom constraint registration procedures for the specified PDK.

Examples

Applies custom PDK settings and retrieves a list of custom constraint registration procedures for the specified PDK.

```
apCustomPDKSettings("CustomPDK" ?regFile "filePath")
=> t
apPDKSetupGetRegProcs("CustomPDK")
=> ("proc1" "proc2" "proc3")
```

apPDKSetupGetVtNameList

```
apPDKSetupGetVtNameList(
     t_pdkName
     )
     => l_procList / nil
```

Description

(Virtuoso Layout Suite EXL) Searches for PDK-specific voltage threshold (VT) CDF parameter names and cell names registered to a particular PDK in the automated device placement and routing flow.

Arguments

t_pdkName

Name of the PDK for which the registered VT CDF parameter names are to be retrieved.

Value Returned

l_procList	A list of strings. The first string is the VT CDF parameter name followed by the cell names.
nil	There are no registered VT CDF parameters for the specified PDK.

Examples

Retrieves a list of VT CDF parameter names and their cell names that are registered to the specified PDK.

```
apPDKSetupGetVtNameList(myPDK)
```

Related Topics

apPlaceAuto

```
apPlaceAuto(
     d_cellviewID
     [ ?fixedAR f_fixedAR ]
     [ ?fixedW f_fixedW ]
     [ ?areaCost f_areaCost ]
     [ ?wireLenCost f_wireLenCost ]
     [ ?symAxis t_symAxis ]
     [ ?fillPopup g_fillPopup ]
     [ ?pOverN g_pOverN ]
     [ ?adjustBoundary g_adjustBoundary ]
     [ ?incr g_incr ]
     [ ?fillPopup g_fillPopup ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Runs the Virtuoso device-level automatic placer with the specified parameters.

Arguments

d_cellviewID Database ID of the cellview.

?fixedAR *f_fixedAR*

Specifies the desired aspect ratio for the generated boundary.

?fixedW *f_fixedW*

Specifies the width of the boundary.

?areaCost *f_areaCost*

Specifies the area cost, depending on which the placer attempts to achieve a compact placement.

?wireLenCost *f_wireLenCost*

Specifies the wire length cost, depending on which the placer attempts to achieve an optimized placement.

?symAxis *t_symAxis*

Specifies the symmetry axis to be used when running the placer.

?fillPopup *g_fillPopup*

Displays a popup window to specify whether existing fill are to be removed.

`?pOverN` *`g_pOverN`*

Specifies whether P-devices can be placed over N-devices.

`?adjustBoundary` *`g_adjustBoundary`*

Specifies whether the PR boundary can be adjusted during device placement.

`?incr` *`g_incr`*

Specifies whether the placer must be run in the incremental mode. When set to `t`, the incremental placer runs without changing the aspect ratio of the figGroups and Modgens in the design.

`?fillPopup` *`g_fillPopup`*

Specifies whether a pop-up window, which prompts if the existing fill are to be removed, is displayed. This argument is honored only when there are base layer fill in the design.

When set to `t` (default), the popup window is displayed with the following options:

- *Delete*: Deletes all base layer fill and runs the placer.
- *Cancel*: Terminates the placer.

When set to `nil`, the popup window is not displayed. All base layer fill are deleted and the placer is run.

Value Returned

`t`	The Virtuoso device-level automatic placer was run.
`nil`	The command was unsuccessful.

Examples

Runs the Virtuoso device-level automatic placer on the specified cellview with default placement settings.

```
; default run
apPlaceAuto(cv)
```

Runs the Virtuoso device-level automatic placer on the specified cellview. A placement boundary as per the specified aspect ratio is generated.

```
; specify aspect ratio
apPlaceAuto(cv ?fixedAR 2.0)
```

Runs the Virtuoso device-level automatic placer on the specified cellview. A placement boundary as per the specified width and wire length is generated. P-devices are not placed over N-devices.

```
; specify width, wireLenCost and turn off pOverN
apPlaceAuto(cv ?fixedW 20 ?wireLenCost 0 ?pOverN nil)
```

Related Topics

apPlaceAutoMatureNode

Auto P&R Assistant User Interface for Device-Level Placement

apPlaceAutoMatureNode

```
apPlaceAutoMatureNode(
     d_cellviewID
     x_effortLevel
     g_updatePRBoundary
     g_incremental
     )
     => t / nil
```

Description

Runs the Virtuoso Analog Placer available at mature nodes.

Arguments

d_cellviewID Database ID of the cellview.

x_effortLevel

Specifies the effort level for running Analog Placer. The valid values are:

- `1`: Minimal effort level
- `2`: Nominal effort level
- `3`: Maximal effort level

The runtime of the placer increases in roughly linear increments as the effort level increases. For example, if a design takes five minutes to place at minimal effort level, then it might take roughly ten minutes at nominal effort level and fifteen minutes at maximal effort level.

g_updatePRBoundary

Specifies whether the PR boundary can be updated during device placement.

g_incremental

Specifies whether the placer must be run in incremental mode. The current design placement is considered as the starting point for running incremental placement.

Value Returned

`t`	The Virtuoso analog placer was run.
`nil`	The command was unsuccessful.

Examples

Run analog placer in incremental mode to fix post placement interactive hand-made adjustments.

```
apPlaceAutoMatureNode(geGetEditCellView() 2 nil t)
```

Related Topics

apPlaceAuto

Auto P&R Assistant User Interface for Device-Level Placement

apResetPDKSetup

```
apResetPDKSetup(
     t_pdkName
     [ ?paramList g_paramsList | ?paramFunc g_paramFunc ]
     [ ?cdsEnvs g_cdsEnvars ]
     [ ?regFile g_regFile ]
     [ ?fillParams g_fillParams ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Unregisters instance parameters, environment variables, custom constraint registration procedures, fill instances parameters, or instance parameter function from the specified PDK.

Arguments

t_pdkName

Name of the PDK from which the specified arguments are to be unregistered.

?paramList *g_paramsList*

Unregisters all master cellviews names and their CDF parameters in the specified PDK.

Either specify ?paramList or ?paramFunc. If both are specified, ?paramFunc is honored.

?paramFunc *g_paramFunc*

Unregisters the function that returns a list of CDF parameter key-value pairs.

?cdsEnvs *g_cdsEnvars*

Unregisters all environment variables in the specified PDK.

?regFile *g_regFile*

Unregisters all registered procedures in the specified PDK.

?fillParams *g_fillParams*

Unregisters all masters and their fill parameters in the specified PDK.

Value Returned

`t`	The specified arguments were unregistered from the specified PDK.
`nil`	There are no registered custom constraint registration procedures for the specified PDK.

Examples

Unregisters all master cellviews names and their CDF parameters in the specified PDK.

```
apResetPDKSetup("customPDK"
                ?paramList t)
```

Unregisters all master cellviews names, their CDF parameters, and CDS environment variables in the specified PDK.

```
apResetPDKSetup("customPDK"
                ?paramList t
                ?cdsEnvs   t)
```

Unregisters all master cellviews names and their CDF parameters, CDS environment variables, procedures, and fill parameters in the specified PDK.

```
apResetPDKSetup("customPDK"
                ?paramList  t
                ?cdsEnvs    t
                ?regFile    t
                ?fillParams t)
```

Unregisters all CDS environment variables, procedures, fill parameters, and the function that returns a list of CDF parameter key-value pairs in the specified PDK.

```
apResetPDKSetup("customPDK"
                ?cdsEnvs    t
                ?regFile    t
                ?paramFunc t
                ?fillParams t)
```

apRunBackannotate

```
apRunBackannotate(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Runs back annotate on the specified cellview for the specified placement style or engine. The function honors the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview for running back annotate.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	Back annotation was run on the given cellview.
`nil`	The command was unsuccessful.

Examples

Uses the options in the Auto P&R Assistant for device style for running back annotate.

```
cv = geGetEditCellView()
apRunBackannotate(cv 'device 'ap)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

apRunCreateGR

```
apRunCreateGR(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Runs the create guard ring command on the specified layout. The function honors the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview on which the command is to be run.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	Guard rings were created.
`nil`	The command was unsuccessful.

Examples

Uses the options in the Auto P&R Assistant for device style for running the create guard rings command.

```
cv = geGetEditCellView()

apRunCreateGR(cv 'device 'ap)
```

Related Topics

apRunDeleteGR

Auto P&R Assistant User Interface for Device-Level Placement

apRunDeleteGR

```
apRunDeleteGR(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Runs the delete guard ring command on the specified layout. The function honors the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview on which the command is to be run.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	Guard rings were deleted.
`nil`	Guard rings could not be deleted.

Examples

Uses the options in the Auto P&R Assistant for device style for running the delete guard rings command.

```
cv = geGetEditCellView()

apRunDeleteGR(cv 'device 'ap)
```

Related Topics

apRunCreateGR

Auto P&R Assistant User Interface for Device-Level Placement

apRunDeviceGeneration

```
apRunDeviceGeneration(
     [ d_cv ]
     [ S_placementStyle ]
     [ S_engineName ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL and above) Generates the selected object types from the source. The function honors the options specified either in the Auto P&R Assistant or through the vreSetOptions SKILL function.

Arguments

d_cv	Specifies cellview on which the command is to be run.
S_placementStyle	Specifies the placement style as either `Device` or `Standard Cell`.
S_engineName	Specifies the name of the placement engine, for example, `ap`.

Value Returned

`t`	The selected object types were generated from the source.
`nil`	Guard rings could not be deleted.

Examples

Uses the options in the Auto P&R Assistant for device style for generating objects.

```
cv = geGetEditCellView()
apRunDeviceGeneration(cv 'device "ap")
```

Runs the command in batch mode.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'device 'ap)
vreSetOption(handle 'init_generateObjects t)
vreSetOption(handle 'init_generateInstances t)
```

```
vreSetOption(handle 'init_generateBoundary t)
apRunDeviceGeneration(cv 'device 'ap)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

apSnapInsts

```
apSnapInsts(
     ?cellView d_cellviewID
     ?snapObjSet l_snapObj
     ?snapToDirX { low | center | high }
     ?snapToDirY { low | center | high }
     [ ?disableRowSnap g_disableRowSnap ]
     [ ?disableSPSnap g_disableSPSNap ]
     [ ?enableWSPSnap g_enableWSPSnap ]
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Snaps the specified instances to rows and snap grids.

Arguments

`?cellView` *`d_cellviewID`*

Database ID of the cellView that contains the instances to be snapped. The defaults is the current cellview.

`?snapObjSet` *`l_snapObj`*

A list of instances to snap. If the set is not specified, all instances in the design are subject to snap.

`?snapToDirX { low | center | high }`

Snap location for the X direction. Valid values are: `low`, `center`, and `high`. The default value is `center`, which is the closet to the grid.

`?snapToDirY { low | center | high }`

Snap location for the Y direction. Valid values are: `low`, `center`, and `high`. The default value is `center`, which is the closet to the grid.

`?disableRowSnap` *`g_disableRowSnap`*

Disable snapping to rows. The default value is `nil`.

`?disableSPSnap` *`g_disableSPSNap`*

Disable snapping to snap grids. The default value is `nil`.

`?enableWSPSnap` *`g_enableWSPSnap`*

Enable snapping to WSP grid. The default value is `nil`.

Value Returned

`t`	Instances were snapped to the specified rows and snap grids.
`nil`	The command was unsuccessful.

Examples

Snaps the selected instances in the current cellview to the specified grids in the X and Y directions.

```
apSnapInsts(
    ?cellView geGetEditCellView()
    ?snapObjSet geGetSelSet()
    ?snapToDirX "low"
    ?snapToDirY "center"
)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

apSwitchPlacerType

```
apSwitchPlacerType(
     S_placementStyle
     [ d_sessionWindow ]
     )
     => t / nil
```

Description

Switches to the specified placer type for the automated placement and routing flow.

Arguments

S_placementStyle

Sets the placement style to either `device` or `stdCell`.

d_sessionWindow

Specifies an optional session window for running the commad.

Value Returned

`t`	The placer type has been switched.
`nil`	The command was unsuccessful.

Examples

Switches placer type to device:

```
apSwitchPlacerType("device")
```

Switches placer type to standard cell:

```
apSwitchPlacerType("stdCell")
```

apUnhighlightDevices

```
apUnhighlightDevices(
     d_cellViewID
     [ ?dummy g_dummies ]
     [ ?active g_activeDevices ]
     )
     => t / nil
```

Description

Unhighlights the selected types of devices in the layout. You can unhighlight dummy devices, active devices, or both.

Arguments

d_cellViewID

Database ID for the cellview in which devices are to be unhighlighted.

?dummy *g_dummies*

Specifies whether dummy devices are to be unhighlighted in the layout.

?active *g_activeDevices*

Specifies whether active devices are to be unhighlighted in the layout.

Value Returned

t	The specified devices were unhighlighted.
nil	The command was unsuccessful.

Examples

Unhighlights all dummy devices in the current cellview.

```
cv = geGetEditCellView()
apUnhighlightDevices(cv ?dummy t ?active nil)
```

Related Topics

apHighlightDevices

apValidateTrims

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

Description

Validates trim insertion in the current design.

Arguments

None

Value Returned

`t`	Trim insertion is correct.
`nil`	Trim insertion is incorrect.

Examples

Validates trim insertion.

```
apValidateTrims()
```

Related Topics

apInsertTrims

apDeleteTrims

Auto P&R Assistant User Interface for Device-Level Placement

lobAddCopyFill

```
lobAddCopyFill(
     d_cellviewID
     l_source
     t_targetArea
     l_netName
     t_createAsMode
     g_hierarchy
     )
     => l_fillFigIDs / nil
```

Description

(Virtuoso Layout Suite EXL) Identifies gaps in the row regions in the given target area and fills them with the specified target fill cells.

Arguments

d_cellviewID	Database ID of the cellview.
l_source	Specifies the instance to be considered as the master for creating fill. Valid values are `Neighbor`, `dbInstId`, `dbCellViewId`, `LCVSpec`, list of `dbInstIds`, list `dbCellViewIds`, list of `LCVSpecs`. Format for LCVSpec: `LCVSpec = list(libName cellname viewName list(list(paramName1 paramValue1) .... list(paramNameN paramValueN)))` Lists are not valid when `createAsMode` is set to `Single`.
t_targetArea	Specifies the target area to be filled. Valid values are cellview ID, name of row region name, or coordinates defining a custom area.
l_netName	Specifies the net names to which the fill must be connected. Default value is "", which indicates no connection.
t_createAsMode	Specifies whether fill must be inserted as a multi-fingered single instance (`Single`) or as single-fingered multiple instances (`Multiple`).
g_hierarchy	Specifies whether copy fill must look through the hierarchy to identify gaps that need to be filled.

Value Returned

l_fillFigs	List of fill IDs that were created.
nil	The command was unsuccessful.

Examples

Fills all the rows within a cellview using the devices adjacent to the identified gap.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    fillDevices=lobAddCopyFill(cellView "Neighbor" cellView~>bBox "" "Single" nil)
  )
)
```

lobIsCopyFill

```
lobIsCopyFill(
     d_dbID
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Checks whether the specified object has been created using the copy fill functions.

Arguments

d_dbID	Database ID of the object.

Value Returned

t	The object was created using copy fill functions.
nil	The object was not created using copy fill functions.

Examples

Selects all copy fill instances that exist within a cellview and checks whether they were created using the copy fill functions.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
     geSelectFigs(setof(inst cellView~>instances lobIsCopyFill(inst))
  )
)
```

lobRemoveCopyFill

```
lobRemoveCopyFill(
     d_cellviewID
     )
     => t / nil
```

Description

(Virtuoso Layout Suite EXL) Removes all objects copy that were created using the copy fill functions from the specified cellview.

Arguments

d_cellviewID	Database ID of the cellview.

Value Returned

t	One or more objects were deleted.
nil	There are no objects created using copy fill functions in the specified cellview.

Examples

Removes all copy fill objects from the current cellview.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    lobRemoveCopyFill(cellView)
```

vcrRemoveShorts

```
vcrRemoveShorts(
     [ ?cv d_cvId ]
     [ selNets l_selNets]
     ) ;
     => t
```

Description

Removes shorts on a net or nets from the layout design. This SKILL API requires Layout MXL license.

Arguments

`?cv` *`d_cvId`*	Database ID of the cellview from which the shorts need to be removed.
`?selNets` *`l_selNets`*	A list of selected nets. If not specified, the shorts are removed from the whole design cell view.

Value Returned

`t`	Always return `t` whether or not there are shorts in the cellview.

Examples

The following example removes all the shorts from the entire cellview.

```
(vcrRemoveShorts (geGetEditCellView))
```

The following example removes the shorts from net10 and net12.

```
(vcrRemoveShorts (geGetEditCellView) (list "net10" "net12"))
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreCapturePlacement

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

Description

Captures placement from the current layout view.

Arguments

None

Value Returned

`t`	Placement was captured.
`nil`	The command was unsuccessful.

Examples

The following example captures placement from the active layout view.

```
vreCapturePlacement()
=> t
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreCreateRowRegion

```
vreCreateRowRegion(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => d_rowRegionId / nil
```

Description

Creates a new row region for the specified scope of the given style or engine using the options specified in the Auto P&R assistant GUI or set by vreSetOption.

Arguments

d_cvid	Database ID of the cellview for which the row region is to be created. The database can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview open in the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active the selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from the active selection on the Auto P&R assistant.

Value Returned

d_rowRegionId	Database ID of the created row region.
`nil`	No row region is created.

Examples

The following example creates a row region for the current cellview in the Auto P&R GUI for `stdCell` style.

```
cv = geGetEditCellView()
vreCreateRowRegion(cv "stdCell" "gp")
```

The following example creates a row region for the current cellview in batch mode for `stdCell` style.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'stdCell 'gp)
vreSetOption(handle 'setup_rowCreation "Create row region")
vreCreateRowRegion(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreDeletePhysicalCells

```
vreDeletePhysicalCells(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Deletes physical-only instances for specified cell types. It uses the options specified in the Auto P&R assistant GUI or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview for which the row region is to be created. The database can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview open in the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from the active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active the selection on the Auto P&R assistant.

Value Returned

`t`	Deletion was successful.
`nil`	Deletion was unsuccessful.

Examples

The following example deletes physical cells for the current cellview in the Auto P&R GUI for `stdCell` style.

```
cv = geGetEditCellView()
vreDeletePhysicalCells(cv "stdCell" "gp")
```

The following example deletes physical cells for the current cellview in batch mode for `stdCell` style.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'stdCell 'gp)
vreSetOption(handle 'place_deleteBoundaryCells t)
vreSetOption(handle 'place_deleteTapCells t)
vreDeletePhysicalCells(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreDeletePowerRouting

```
vreDeletePowerRouting(
     [ d_cvid ]
     [ S_rStyle ]
     [ S_rName ]
     )
     => t / nil
```

Description

Deletes power routing data for the specified scope of the given routing style or router using the options specified in the Routing Assistant or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview of which routing needs to be deleted. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview open in the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing information is deleted.
`nil`	Routing information could not be deleted.

Examples

The following example deletes power routing when options have already been specified in the Routing assistant UI for the standard cell style routing.

```
cv = geGetEditCellView()
vreDeletePowerRouting(cv "stdCell" "nr")
```

The following example specifies three options in batch mode for the device-level routing, and then deletes power routing information.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil supply_nets "All" supply_netsWithin "PR boundary"
supply_deleteStripes t supply_deleteVias t))
vreDeletePowerRouting(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreDeleteRowRegions

```
vreDeleteRowRegions(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Deletes all automatically created rows and row regions using the options specified in the Auto P&R assistant GUI or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview from which rows and row regions are to be deleted. The database can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview open in the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from the active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from the active selection on the Auto P&R assistant.

Value Returned

`t`	Deletion was successful.
`nil`	Deletion was unsuccessful.

Examples

The following example deletes rows and row regions from the current cellview in the Auto P&R GUI for stdCell style.

```
cv = geGetEditCellView()
vreDeleteRowRegions(cv "stdCell" "gp")
```

The following example deletes rows and row regions from the current cellview in batch mode for `stdCell` style.

```
cv = geGetEditCellView()
vreDeleteRowRegions(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreDeleteSignalRouting

```
vreDeleteSignalRouting(
     [ d_cvid ]
     [ S_rStyle ]
     [ S_rName ]
     )
     => t / nil
```

Description

Deletes signal routing data for specified scope based of the given routing style or router using the options specified in the Routing Assistant or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview of which routing needs to be deleted. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing information is deleted.
`nil`	Routing information could not be deleted.

Examples

The following example deletes signal routing when options have already been specified in the Routing assistant UI for the standard cell style routing.

```
cv = geGetEditCellView()
vreDeleteSignalRouting(cv "stdCell" "nr")
```

The following example specifies options in batch mode for the device-level routing, and then performs signal routing deletion.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current Cellview"))
vreDeleteSignalRouting(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreFinishRouting

```
vreFinishRouting(
     [ d_cvid ]
     [ S_rStyle ]
     [ S_rName ]
     )
     => l_result / nil
```

Description

Runs finish routing based on the given routing style or router using the options specified in the Routing Assistant or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview on which finish routing needs to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

l_result	Disembodied property list representing statistics for the finished routing.
`nil`	The routing was not run successfully.

Examples

The following example runs finish routing on the specified cellview using the specified router and routing style and the routing options specified in the Routing assistant are considered.

```
cv = geGetEditCellView()
vreFinishRouting(cv "device" "gbr")
```

The following example specifies options in batch mode for the device-level routing and runs finish routing.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_deleteWires t route_deleteVias t))
vreFinishRouting(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGenerateWSPs

```
vreGenerateWSPs(
     [ d_cvid ]
     [ S_rStyle ]
     [ S_rName ]
     )
     => d_wsspDefId / nil
```

Description

Generates WSPs for the selected cellview. This SKILL API uses the options specified in the Routing Assistant or through `vreSetOption` based on the given routing style or router.

Arguments

d_cvid	Database ID of the cellview for generating WSPs. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
s_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
s_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

d_wsspDefId	Database ID of the created WSSPDef.
`nil`	WSP has not been created.

Examples

The following example generates WSPs when options have already been specified in the Routing assistant for the standard cell routing style.

```
cv = geGetEditCellView()
vreGenerateWSPs(cv "stdCell" "nr")
```

The following example specifies options in batch mode for the device-level routing and then generate WSPs.

```
cv = geGetEditCellView()
```

```
layerData = '((nil generate t layer "M2" spacing 0.05 width 0.03)
              (nil generate t layer "M4" spacing 0.05 width 0.03))
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions (handle 'setup_WSPLayers layerData)
vreGenerateWSPs(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetCellView

```
vreGetCellView(
     g_vreHandle
     )
     => d_cvId
```

Description

Returns the cellview associated with the supplied routing environment handle object.

Arguments

g_vreHandle	Routing environment handle object for which the associated cellview is required.

Value Returned

d_cvId	Database ID of the cellview associated with the supplied routing environment handle object.

Examples

Returns the database ID of the cellview associated with the specified routing environment handle object.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreGetCellView(handle)
=> cv
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetHandle

```
vreGetHandle(
     d_cvid
     s_routingStyle
     s_router
     )
     => g_vreHandle / nil
```

Description

Creates a routing environment handle object which captures the cellview, routing style, and router. This object can then be passed to other routing environment APIs that take a `vreHandle` parameter.

Arguments

d_cvid	Database ID of the cellview for which the handle object is to be created.
s_routingStyle	Routing style to be captured in the handle object.
s_router	Router to be captured in the handle object.

Value Returned

g_vreHandle	Routing environment handle object created for the specified cellview, routing style, and router.
`nil`	The routing environment handle object could not be created.

Examples

The following example creates a routing environment handle object for the current cellview with device-level routing using the global router.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetOption

```
vreGetOption(
     g_vreHandle
     s_optionName
     )
     => g_optionValue / nil
```

Description

Returns the value of the specified option for the supplied routing environment handle object.

Arguments

g_vreHandle	Routing environment handle object from which an option value is to be retrieved.
s_optionName	Symbol for the option whose value is required. The list of available option names can be retrieved by using `(vreGetOptions handle)->?`

Value Returned

g_optionValue	Value of the specified option in the specified handle object.
`nil`	The option value could not be determined.

Examples

Returns the value of the `'supply_nets` option in the specified routing environment handle object.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreGetOption(handle 'supply_nets)
=> "All"
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetOptions

```
vreGetOptions(
     g_vreHandle
     )
     => g_optionsList / nil
```

Description

Returns a disembodied property list of all the options and values associated with the supplied routing environment handle object.

Arguments

g_vreHandle	Routing environment handle object from which option values are to be retrieved.

Value Returned

g_optionsList	List of options and values associated with the specified handle.
`nil`	The list of options could not be returned.

Examples

The following example returns the list of options for the specified routing environment handle object. From that list, the values of individual options are queried and the `'supply_nets` option is updated from `All` to `Selected`.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
options=vreGetOptions(handle)
 => <DPL too long to show>

;; Iterate over options
foreach( optionName options->?
   optionValue = (get options optionValue)
;; Do something with optionName and optionValue
)

;; Get an individual option
options->supply_nets
 => "All"

;; Update an option value in the DPL (DPL must be
;; passed to vreSetOptions to actually set options)
```

```
options->supply_nets = "Selected"
 => "Selected"
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetRouter

```
vreGetRouter(
     g_vreHandle
     )
     => s_router / nil
```

Description

Returns the symbol for the router associated with the supplied routing environment handle object.

Arguments

g_vreHandle	Routing environment handle object for which the router symbol is to be retrieved.

Value Returned

s_router	Symbol for the router associated with the specified handle object.
nil	The router symbol could not be returned.

Examples

In the following example, the `vreHandle` object is being created as a consolidated representation of the input arguments to `vreGetHandle()`, cellview, routing style, and router. The `vreGetRouter()` is then called on the same `vreHandle` object to return the router symbol that was originally used to create it.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreGetRouter(handle)
 => 'gbr
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetRoutingStyle

```
vreGetRoutingStyle(
     g_vreHandle
     )
     => s_routingStyle
```

Description

Returns the routing style associated with the supplied routing environment handle object.

Arguments

g_vreHandle	Routing environment handle object for which the routing style is to be retrieved.

Value Returned

s_routingStyle	Symbol for the routing style associated with the specified handle object.

Examples

Returns the routing style symbol for the specified routing environment handle object.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreGetRoutingStyle(handle)
 => 'device
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreIsOption

```
vreIsOption(
     g_vreHandle
     s_optionName
     )
     => t / nil
```

Description

Indicates whether a named option is valid for the specified `vreHandle` before calling `vreGetOption` or `vreSetOption`. Directly manipulating an option value that is not valid for the specified `vreHandle` using `vreGetOption` or `vreSetOption` results in a warning message.

Arguments

g_vreHandle	Routing environment handle object that is to be queried.
s_optionName	Symbol for the option whose validity is being tested with respect to the specified handle.

Value Returned

`t`	The specified option is valid for this handle and can be retrieved using `vreGetOption` and set using `vreSetOption`.
`nil`	The specified option is invalid for this handle. Attempting to retrieve this using `vreGetOption` or set it using `vreSetOption` results in a warning.

Examples

Indicates whether some options are valid for a given `vreHandle`.

```
handle = (vreGetHandle (geGetEditCellView) 'device 'gbr)
vreIsOption (handle 'route_nets)
=> t
vreIsOption (handle 'notAnOption)
=> nil
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreGetOption

vreGetHandle

vreSetOption

vreLoadPreset

```
vreLoadPreset(
     t_presetName
     g_vreHandle
     )
     => t / nil
```

Description

Loads a router preset file.

Arguments

t_presetName	Name of the preset file that is to be loaded.
g_vreHandle	Routing environment handle object for which the option value is to be set.

Value Returned

t	The preset file is loaded.
nil	The specified preset file is invalid for this handle.

Examples

Loads the default preset file.

```
vreLoadPreset("Defaults")
==> t
handle = vreGetHandle(cv 'stdCell 'nr)
vreLoadPreset("nrPresets" ?handle handle)
==> t
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRaiseConstraintManager

```
vreRaiseConstraintManager(
     [ d_cvid ]
     [ S_rStyle ]
     [ S_rName ]
     )
     => t / nil
```

Description

Opens the Routing Constraint Manager based on the given routing style or router. This API uses the options specified in the Routing Assistant or set through `vreSetOption` based on the given routing style or router.

Arguments

d_cvid	Database ID of the cellview for which Routing Constraint Manager should be opened. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing Constraint Manager opens.
`nil`	Routing Constraint Manager could not be opened.

Examples

The following example brings up the Routing Constraint Manager window for the current cellview and the standard cell routing style using string type arguments.

```
vreRaiseConstraintManager(geGetEditCellView() "stdCell" "nr")
```

The following example brings up the Routing Constraint Manager window for the current cellview and the device level routing style using symbol type arguments.

```
vreRaiseConstraintManager(geGetEditCellView() 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRaisePreRoutingBrowser

```
vreRaisePreRoutingBrowser(
     [ d_cvid ]
     [ S_rStyle ]
     [ S_rName ]
     )
     => t / nil
```

Description

Opens the Pre-routing Browser based on the given routing style and router name.

Arguments

d_cvid	Database ID of the cellview for opening pre routing browser. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Pre-routing Browser opens successfully.
`nil`	Pre-routing Browser does not open.

Examples

The following examples open pre-routing browser.

```
vreRaisePreRoutingBrowser(geGetEditCellView() "stdCell" "nr")
vreRaisePreRoutingBrowser(geGetEditCellView() 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRaiseResultsBrowser

```
vreRaiseResultsBrowser(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Opens the Routing Results Browser for the specified scope. This API uses the options specified in the Routing Assistant or set through `vreSetOption` based on the given routing style or router.

Arguments

d_cvid	Database ID of the cellview for which Routing Results Browser should be opened. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing Results Browser opens.
`nil`	Routing Results Browser could not be opened.

Examples

The following example displays the Routing Results Browser for the standard cell routing style.

```
cv = geGetEditCellView()
vreRaiseResultsBrowser(cv "stdCell" "nr")
```

The following example brings up the Routing Results Browser for the device level routing style.

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil results_nets "All" results_supplyNets t))
vreRaiseResultsBrowser(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRemoveJogs

```
vreRemoveJogs(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Removes jogs from the given cellview when the specified router is run on it. This API uses the options specified in the Routing Assistant or set through `vreSetOption` based on the given routing style or router.

Arguments

d_cvid	Database ID of the cellview on which routing is run to remove jogs. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	The jogs were removed.
`nil`	Jogs could not be removed.

Examples

The following example will remove jogs for standard cell routing style.

```
cv = geGetEditCellView()
vreRemoveJogs(cv "device" "gbr")
```

The following example will remove jogs in batch mode for device level routing style.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current cellview"))
vreRemoveJogs(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRemoveNotches

```
vreRemoveNotches(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

'Removes notches from the given cellview when the specified router is run on it. This API uses the options specified in the Routing Assistant or set through `vreSetOption` based on the given routing style or router.

Arguments

d_cvid	Database ID of the cellview on which routing is run to remove notches. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	The notches were removed.
`nil`	Notches could not be removed.

Examples

The following example will remove notches for the standard cell routing style.

```
cv = geGetEditCellView()
vreRemoveNotches(cv "device" "gbr")
```

The following example will remove notches in batch mode for the device level routing style.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current cellview"))
vreRemoveNotches(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunAssistedRouter

```
vreRunAssistedRouter(
     d_cvid
     S_rStyle
     S_rName
     )
     => t / nil
```

Description

Runs assisted routing for the specified scope using the options specified in the Routing Assistant or set using `vreSetOption` based on the given routing style and router.

Arguments

d_cvid	Database ID of the cellview on which assisted routing is to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing was run successfully.
`nil`	Routing was unsuccessful.

Examples

The following example runs the assisted router using the standard cell style routing based on the options set in the Routing Assistant.

```
cv = geGetEditCellView()
vreRunAssistedRouter(cv "device" "gbr")
```

The following example sets three options in batch mode before running the device-level routing style:

```
cv = geGetEditCellView()
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current cellview"))
vreRunAssistedRouter(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunChecker

```
vreRunChecker(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Runs the pre-route checker for the specified scope using the options specified in the Routing Assistant or set using `vreSetOption` based on the given routing style and router.

Arguments

d_cvid	Database ID of the cellview of which pre-route checks are to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Pre-route checks completed.
`nil`	Pre-route checks were unsuccessful.

Examples

The following example runs the pre-route checker when options have already been specified in the Routing Assistant UI for standard cell routing style.

```
cv = geGetEditCellView()
vreRunChecker(cv "stdCell" "nr")
```

The following example specifies options in batch mode for device-level routing style and then runs pre-route checker.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil check_existingDRCs nil check_displayLog t
check_overwriteLog t check_generateMarkers nil))
vreRunChecker(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunGeneration

```
vreRunGeneration(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Generates selected object types from source. It uses the options specified in the Auto P&R assistant or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview to be used for generation of object types. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active selection on the Auto P&R assistant.

Value Returned

`t`	Generation was successful.
`nil`	Generation was unsuccessful.

Examples

The following example generates objects in the Auto P&R GUI for `stdCell` style.

```
cv = geGetEditCellView()
vreRunGeneration(cv 'stdCell 'gp)
```

The following example generates objects in batch mode for `stdCell` style.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'stdCell 'gp)
vreSetOption(handle 'init_generateObjects t)
vreSetOption(handle 'init_generateInstances t)
vreSetOption(handle 'init_generateBoundary t)
vreRunGeneration(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunMeshRouter

```
vreRunMeshRouter(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Runs the mesh router for the specified scope using the options specified in the Routing Assistant or set using `vreSetOption` based on the given routing style and router.

Arguments

d_cvid	Database ID of the cellview on which mesh routing is to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing was run successfully.
`nil`	Routing was unsuccessful.

Examples

The following example performs mesh routing when options have already been specified in the Routing Assistant UI for standard cell routing style.

```
cv = geGetEditCellView()
vreRunMeshRouter(cv "device" "gbr")
```

The following example specifies options in batch mode for device-level routing and then performs mesh routing.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current cellview"))
vreRunMeshRouter(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunP2TRouter

```
vreRunP2TRouter(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Runs the Pin to Trunk router for the specified scope using the options specified in the Routing Assistant or set using `vreSetOption` based on the given routing style and router.

Arguments

d_cvid	Database ID of the cellview on which Pin to Trunk routing is to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing was run successfully.
`nil`	Routing was unsuccessful.

Examples

The following example runs the Pin to Trunk router using standard cell style routing based on the options set in the Routing Assistant.

```
cv = geGetEditCellView()
vreRunP2TRouter(cv "device" "gbr")
```

The following example sets three options in batch mode before running device-level Pin to Trunk routing:

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil route_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current cellview"))
vreRunP2TRouter(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunPlacer

```
vreRunPlacer(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Runs the specified stages of the placement flow using the options specified in the Auto P&R assistant or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview for placement. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active selection on the Auto P&R assistant.

Value Returned

`t`	Placement was successful.
`nil`	Placement was unsuccessful.

Examples

The following example runs placement in the Auto P&R GUI for `stdCell` style.

```
cv = geGetEditCellView()
vreRunPlacer(cv 'stdCell 'gp)
```

The following example runs placement in batch mode for `stdCell` style.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'stdCell 'gp)
vreSetOption(handle 'place_placeStdCells t)
vreRunPlacer(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunPowerRouter

```
vreRunPowerRouter(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Runs the power router for the specified scope using the options specified in the Routing Assistant or set using `vreSetOption` based on the given routing style and router.

Arguments

d_cvid	Database ID of the cellview on which power routing is to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing was run successfully.
`nil`	Routing was unsuccessful.

Examples

The following example performs power routing when options have already been specified in the Routing Assistant UI for standard cell routing style.

```
cv = geGetEditCellView()
vreRunPowerRouter(cv "stdCell" "nr")
```

The following example specifies options in batch mode for device-level routing and then performs power routing.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil supply_nets "All" supply_netsWithin "PR boundary"
supply_routedLoc "Current cellview"))
vreRunPowerRouter(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreRunSignalRouter

```
vreRunSignalRouter(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Runs the signal router for the specified scope using the options specified in the Routing Assistant or set using `vreSetOption` based on the given routing style and router.

Arguments

d_cvid	Database ID of the cellview on which signal routing is to be run. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing was run successfully.
`nil`	Routing was unsuccessful.

Examples

The following example performs signal routing when options have already been specified in the Routing Assistant UI for standard cell routing style.

```
cv = geGetEditCellView()
vreRunSignalRouter(cv "stdCell" "nr")
```

The following example specifies options in batch mode for device-level routing and then performs signal routing.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil signal_nets "All" route_netsWithin "PR boundary"
route_routedLoc "Current cellview"))
vreRunSignalRouter(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreSetOption

```
vreSetOption(
     g_vreHandle
     s_optionName
     g_optionValue
     )
     => t / nil
```

Description

Sets the value of the specified option for the supplied routing environment handle object.

Arguments

g_vreHandle	Routing environment handle object for which the option value is to be set.
s_optionName	Name of the option to be set.
g_optionValue	Value to be set for the specified option.

Value Returned

`t`	Specified option was set to the supplied value.
`nil`	Specified option could not be set to the supplied value.

Examples

The following example sets the value of the `'supply_nets` option to `"Selected"`.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreSetOption(handle 'supply_nets "Selected")
 => t
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreSetOptions

```
vreSetOptions(
     g_vreHandle
     g_optionsList
     )
     => t / nil
```

Description

Sets all the options associated with the supplied routing environment handle object to the values in a specified DPL (typically returned by `vreGetOptions()`).

Arguments

g_vreHandle	Routing environment handle object for which the option value is to be set.
g_optionsList	List of options and values associated with the specified handle.

Value Returned

`t`	Options were set as specified.
`nil`	Options could not be set to the specified values.

Examples

Retrieves a list of all the options present on the supplied handle object and then sets the `supply_nets` option to `"Selected"`.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
options=vreGetOptions(handle)
 => <DPL too big to show>
;; Update an option value in the DPL (DPL must be
;; passed to vreSetOptions to actually set options)
options->supply_nets = "Selected"
 => "Selected"
vreSetOptions(handle options)
 => t
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreSetRouter

```
vreSetRouter(
     g_vreHandle
     s_router
     )
     => s_router
```

Description

Updates the supplied routing environment handle object to use the specified router. Only certain combinations of router and routing style are supported.

Arguments

g_vreHandle	Routing environment handle object for which the router is to be set.
s_router	Symbol for the router to be used.

Value Returned

s_router	Router was set for the supplied handle object.

Examples

The following example changes the router for the specified handle object from `'gbr` to `'vcr`. This creates an invalid combination of router and routing style, so the routing style is then updated to `'chip`.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreSetRouter(handle 'vcr)
 => 'vcr
;; Has created an invalid combination of 'device style and 'vcr router, so
;; now switch the routing style to match the router
vreSetRoutingStyle(handle 'chip)
 => 'chip
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreSetRoutingStatus

```
vreSetRoutingStatus(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Sets the specified routing status on nets within the specified scope.

Arguments

d_cvid	Database ID of the cellview for routing objects. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Routing status is set successfully.
`nil`	Setting routing status was unsuccessful.

Examples

The following example sets the routing status when options have already been specified in the Routing Assistant UI for standard cell routing style.

```
cv = geGetEditCellView()
vreSetRoutingStatus(cv "stdCell" "nr")
```

The following example specifies options in batch mode for device-level routing and then sets the routing status.

```
cv = geGetEditCellView()
```

```
handle = vreGetHandle(cv 'device 'gbr)
vreSetOptions(handle '(nil eco_nets "All" eco_netsWithin "PR boundary"
eco_routedLoc "Current cellview" eco_setStatus "Fixed"))
vreSetRoutingStatus(cv 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreSetRoutingStyle

```
vreSetRoutingStyle(
     g_vreHandle
     s_routingStyle
     )
     => s_routingStyle
```

Description

Updates the supplied routing environment handle object to use the specified routing style. Only certain combinations of router and routing style are supported.

Arguments

g_vreHandle	Routing environment handle object for which the routing style is to be set.
S_routingStyle	Symbol for the routing style to be used.

Value Returned

S_routingStyle	Routing style was set for the supplied handle object.

Examples

The following example changes the routing style for the specified handle object from `'device` to `'chip`. This creates an invalid combination of router and routing style, so the router is then updated to `'vcr`.

```
cv=geGetEditCellView()
handle=vreGetHandle(cv 'device 'gbr)
vreSetRoutingStyle(handle 'chip)
 => 'chip
 ;; Has created invalid combination of 'chip style and 'gbr router,
 ;; so switch the router to match
vreSetRouter(handle 'vcr)
 => 'vcr
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreShowCheckerLog

```
vreShowCheckerLog(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Displays the log file viewer for the pre-route checker.

Arguments

d_cvid	Database ID of the cellview for which the checker log is to be shown. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Log file viewer was displayed.
`nil`	Log file viewer could not be displayed.

Examples

```
vreShowCheckerLog(geGetEditCellView() "stdCell" "nr")
vreShowCheckerLog(geGetEditCellView() 'device 'gbr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreShowPlacerLog

```
vreShowPlacerLog(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Displays the placement log window.

Arguments

d_cvid	Database ID of the cellview used for placement. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active selection on the Auto P&R assistant.

Value Returned

t	Placement log window was displayed successfully.
`nil`	Placement log window could not be displayed.

Examples

The following example displays the placement log for the `stdCell` placement style.

```
vreShowPlacerLog(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreShowRowGenLog

```
vreShowRowGenLog(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => d_RowRegionId / nil
```

Description

Displays row generation log based on the scope defined for the given routing style using the options specified in the Auto P&R assistant or set by `vreSetOption`.

Arguments

d_cvid	Database ID of the cellview used for row region creation. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active selection on the Auto P&R assistant.

Value Returned

d_RowRegionId	Database ID of the created `rowRegion`.
`nil`	Placement log window could not be displayed.

Examples

The following example displays the row gen log for the `stdCell` placement style.

```
cv = geGetEditCellView()
vreShowRowGenLog(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreShowSignalRouterLog

```
vreShowSignalRouterLog(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t
```

Description

Displays the log file viewer for the signal type router.

Arguments

d_cvid	Database ID of the cellview for which the log file is to be shown. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Log file viewer was displayed.

Examples

```
vreShowSignalRouterLog(cv 'stdCell 'nr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreStopPlacer

```
vreStopPlacer(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Stops the active placer.

Arguments

d_cvid	Database ID of the cellview for placer. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active selection on the Auto P&R assistant.

Value Returned

t	Placer was successfully stopped.
`nil`	Unable to stop placer or the placer is inactive.

Examples

The following example stops the active placer.

```
vreStopPlacer(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreStopRowRegionCreation

```
vreStopRowRegionCreation(
     [ d_cvid ]
     [ S_style ]
     [ S_eName ]
     )
     => t / nil
```

Description

Stops the active row region generation.

Arguments

d_cvid	Database ID of the cellview for which row region generation is to be stopped. The database can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview open in the current window.
S_style	Specifies the placement style, for example `stdCell`. The default value is taken from active selection on the Auto P&R assistant.
S_eName	Specifies the engine name, for example `gp`. The default value is taken from active selection on the Auto P&R assistant.

Value Returned

`t`	Generation was successfully stopped
`nil`	Unable to stop generation or the generator is inactive.

Examples

The following example stops the row region generation for the specified style and engine.

```
vreStopRowRegionCreation(cv 'stdCell 'gp)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreStopSignalRouter

```
vreStopSignalRouter(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Stops the active signal router.

Arguments

d_cvid	Database ID of the cellview being routed. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	Router was stopped.
`nil`	The router could not be stopped or the router is not active.

Examples

The following example stops the signal router for the specified cellview, routing style, and router.

```
vreStopSignalRouter(cv 'stdCell 'nr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vreStopWSPGeneration

```
vreStopWSPGeneration(
     [ d_cvid ]
     [ S_rStyle]
     [ S_rName ]
     )
     => t / nil
```

Description

Stops the active WSP generation.

Arguments

d_cvid	Database ID of the cellview in which WSPs are being generated. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
S_rStyle	Specifies the routing style, which can be either `device`, `stdCell`, or `chip`.
S_rName	Specifies the router name, which can be either `gbr`, `nr`, or `vcr`.

Value Returned

`t`	WSP generation was stopped.
`nil`	WSP generation could not be stopped or is not active.

Examples

The following example stops WSP generation for the specified cellview, routing style, and router.

```
vreStopWSPGeneration(cv 'stdCell 'nr)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

```
vrtCheckDesign(
     [ d_cvid ]
     [ ?layers l_layers ]
     [ ?region l_region ]
     [ ?all { t | nil } | ?checkDeadEndPins ?checkDeviceTrims ?checkDiffPins
       ?checkGatePins ?checkGridSanity ?checkPinWSPConformance ?checkSDPins
       ?checkSymmetry ?checkTracks ?checkTrimValidity ?checkVias ]
     [ ?showLog | ?logVios | ?createMarkers ]
     )
     => t / nil
```

Description

Runs pre-routing checks either on a whole cellview or on certain layers or a region of the design. You can either select the specific checks to run or run all available checks.

Arguments

d_cvid	Database ID of the cellview to be checked. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
`?layers` *l_layers*	Performs pre-routing checks on the specified list of layers. If no layer list is specified, the checks are run on all routing layers in the cellview.
`?region` *l_region*	The lower-left and upper-right coordinates of the region in which checks are to be performed. If no region is specified, everything inside the PR boundary or cell boundary is checked.

`?all { t | nil }`

Runs all available pre-route checks. Alternatively, specify the individual checks to be run from the following list:

- `?checkDeadEndPins` - Checks for the pins covered by blockages.
- `?checkDeviceTrims` - Checks for the cut shapes on device layers.
- `?checkDiffPins` - Checks the mismatch between diffusion pin net on device and top-level net.
- `?checkGatePins` - Checks for misalignment between WSP tracks and gate pins.
- `?checkGridSanity` - Checks grid sanity.
- `?checkPinWSPConformance` - Checks for misalignment between WSP tracks and top-level pins.
- `?checkSDPins` - Checks for the misalignment of WSP tracks with source and drain pins.
- `?checkSymmetry` - Checks symmetrical nets pin placement.
- `?checkTracks` - Checks the spacing of WSPs on tracks.
- `?checkTrimValidity` - Checks for the insufficient spacing for trim insertion.
- `?checkVias` - Checks for congestion or limitations for via placement.

`?showLog | ?logVios | ?createMarkers`

Specifies how the information obtained from the checks is presented to the user.

- `?showLog` displays a violation log file.
- `?logVios` lists violations in the CIW.
- `?createMarkers` creates violation markers in the design.

Value Returned

`t`	The routing checks completed successfully.
`nil`	The routing checks did not complete due to errors.

Examples

The following example runs all checks on the specified design.

```
vrtCheckDesign(geGetEditCellView() ?all)
```

The following example runs only the dead-end pin check on the current design.

```
vrtCheckDesign(?checkDeadEndPins ?checkSymmetry ?checkDiffPins)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

vrtCreateIDLayer

```
vrtCreateIDLayer(
     [ ?cv d_cvid ]
     [ ?debug { t | nil } ]
     )
     => layer_ID / nil
```

Description

Creates an ID layer according to advanced node `OppExtensionRules`.

Arguments

`?cv` *`d_cvid`*	Database ID of the cellview.
`?debug { t | nil }`	Prints the debug information.

Value Returned

`layer_ID`	Returns ID of the layer.
`nil`	The layer ID is not created.

Examples

```
vrtCreateIDLayer( ?debug t)
=>
Loading gbr.cxt
Loading techComp.cxt
minOppExtension rule("M0.R.30")
"AN_HDU1M0_3T" (0.017000 "vertical")
"M0" (0.112500 "horizontal")
insts bBxo(0.135:0.052 1.305:0.796)
Extended bBxo(0.0225:-0.017 1.4175:1.057)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

vrtDeleteNets

```
vrtDeleteNets(
     d_cvid
     [ ?nets l_nets ]
     [ ?layers l_layers ]
     [ ?region l_region ]
     [ ?options l_options ]
     [ ?logFile g_logFile ]
     [ ?appendToLog g_appendToLog ]
     )
     => t / nil
```

Description

Deletes the routing on the nets. The routing is done by the signal router or power router either on the whole cellview or on certain layers or a region of the design. It can also delete pre-routes of nets that are not generated by signal router or power router.

Arguments

d_cvid	Database ID of the layout cellview for which nets are to be deleted. The database ID can be obtained using the `geGetEditCellView` SKILL function.
`?nets` *l_nets*	Specifies a list of routed nets to be deleted. If no net list is specified, it deletes routing on all the nets in the cellview.
`?layers` *l_layers*	Deletes the routed nets from the specified list of layers. If no layer list is specified, the nets are deleted from all layers in the cellview.
`?region` *l_region*	The lower-left and upper-right coordinates of a region from which nets are to be deleted. If no region is specified, all nets inside the PR boundary or cell boundary are deleted.
`?options` *l_options*	Specifies the name of the objects to be deleted. If no object list is specified, nothing is deleted. The valid object types are: `path`, `via`, `trunk`, `signal`, `preroute`, `powerPath`, and `powerVia`.
`?logFile` *g_logFile*	Specifies the name of a log file in which router messages are captured.

`?appendToLog` *g_appendToLog*

	Specifies that new messages are appended to the log file if it already exists. If set to `nil`, the existing file is overwritten.

Value Returned

`t`	The routed nets were deleted without any errors.
`nil`	The routed nets were not deleted due to errors.

Examples

The following example deletes all preroutes and signal routed pathsegs for nets `n1` and `n2` on layer `M1` and `M2` inside the PR boundary of the design.

```
vrtDeleteNets(geGetEditCellView() '("n1" "n2")
geGetEditCellView()->prBoundary->bBox '("M1" "M2"), '("preroute" "path"))
```

The following example deletes all power routed pathSegs and vias for all the nets in the current design.

```
vrtDeleteNets(geGetEditCellView() nil nil nil '("powerPath" "powerVia"))
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

vrtPowerRoute

```
vrtPowerRoute(
     d_cvid
     [ ?nets l_nets ]
     [ ?layer l_layers ]
     [ ?region l_region ]
     [ ?options l_options ]
     [ ?sharedTrack g_sharedTrack ]
     [ ?net2WireTypes g_net2WireTypes ]
     [ ?pullbacks g_pullbacks ]
     [ ?postFixTrim g_postFixTrim ]
     [ ?postFixMinLength g_postFixMinLength ]
     [ ?postFixEolKeepout g_postFixEolKeepout ]
     [ ?logFile g_logFile ]
     [ ?appenToLog g_appendToLog ]
     )
     => t / nil
```

Description

Runs power routing on the nets in the design.

Arguments

d_cvid	Database ID of the cellview on which power routing is to be performed. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
`?nets` *l_nets*	A list of net names on which power routing is to be performed. If no net list is specified, it routes all the nets in the cellview.
`?region` *l_region*	A list specifying the lower-left and upper-right coordinates of the region where power routing is to be performed. If no region is specified, all nets inside the PR boundary or cell boundary are considered.
`?layers` *l_layers*	A list of layers to perform power routing. If no layer is specified, power routing is performed on all layers.

`?options` *`l_options`*
: List of objects for power routing. If no object list is specified, nothing is routed. The valid object types are: `createPath`, `createVia`, `createPin`, `connectPin`, `createFigGroup`, `sharedTrack`, `postFixTrim`, and `postFixMinLength`.

`?sharedTrack` *`g_sharedTrack`*
: Boolean value to specify whether the power router share a same track to different power net.

`?net2WireTypes` *`g_net2WireTypes`*
: Specifies the wire type location mapping for power nets.

`?postFixTrim` *`g_postFixTrim`*
: Boolean value to specify whether to do a post trim patch fixing after power routing.

`?postFixMinLength` *`g_postFixMinLength`*
: Boolean value to specify whether to do a post minLength drc error fixing after power routing.

`?postFixEolKeepout` *`g_postFixEolKeepout`*
: Boolean value to specify whether to do a post end of line spacing drc error fixing after power routing.

`?logFile` *`g_logFile`*
: Specifies the name of a log file in which router messages are captured.

`?appendToLog` *`g_appendToLog`*
: Specifies that new messages are appended to the log file if it already exists. If set to `nil`, the existing file is overwritten.

Value Returned

`t`	Nets were power routed successfully without any errors.
`nil`	Routing was unsuccessful due to errors.

Examples

The following example runs the Power Route command on all the wires and vias between `M1` and `M2`.

```
vrtPowerRoute(geGetEditCellView() ?layers '("M1" "M2") ?options '("createPath"
"createVia"))
```

The following example inserts vias between `M1` and `M2` on `vss` and `vdd` net.

```
vrtPowerRoute(geGetEditCellView() ?nets '("vss" "vdd") ?layers '("M1" "M2")
?options '("createVia"))
```

The following example runs the Power Route command on all the PG nets to create wires and vias and create a figGroup from `M1` to `M3` inside PR boundary.

```
vrtPowerRoute(geGetEditCellView() ?region?geGetEditCellView()->prBoundary->bBox
?layers '("M1" "M2" "M3") ?options? '("cratePath" "createVia" "createFigGroup")))
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

vrtRouteAssisted

```
vrtRouteAssisted(
     d_cvid
     [ nets l_nets ]
     [ layers l_layers ]
     [ regionBox l_regionBox ]
     [ ?generateSpine g_generateSpine ]
     [ ?spineLayer g_spineLayer ]
     [ ?connectPins g_connectPins ]
     [ ?generateMesh g_generateMesh ]
     [ ?meshBottomLayer t_meshBottomLayer ]
     [ ?meshTopLayer t_meshTopLayer ]
     [ ?meshSetting l_meshSetting ]
     [ ?meshExtendPreRoute g_meshExtendPreRoute ]
     [ ?finishRouting g_finishRouting ]
     [ ?preferredLayerBottom t_preferredLayerBottom ]
     [ ?preferredLayerTop t_preferredLayerTop ]
     [ ?postFixTrim g_postFixTrim ]
     [ ?postFixMinLength g_postFixMinLength ]
     [ ?postFixEolKeepout g_postFixEOLKeepout ]
     [ ?minSkipTrack d_minSkipTrack ]
     [ ?createFigGroup g_createFigGroup ]
     )
     => t / nil
```

Description

If both Generate Mesh and Finish Routing are selected and top and bottom layers are different in mesh control, Finish Routing starts first and then Generate Mesh.However, if the top and bottom layers are the same in mesh control, Generate Spine starts first and then Finish Routing.

In case, top and bottom layers are the same in Generate Mesh, Generate Spine command starts to create spines.

Arguments

d_cvid	Database ID of the layout cellview of which nets are to be routed. The database ID can be obtained using the `geGetEditCellView` SKILL function. The default value is the cellview of the current window.
`nets` *l_nets*	A list of net name to be routed. If no net list is specified, it routes all the nets in the cellview.

`region` *`l_region`*
: A list specifying the lower-left and upper-right coordinates of the region where nets are to be routed. If no region is specified, all nets inside the PR boundary or cell boundary are considered.

`layers` *`l_layers`*
: A list of layer names to be routed. If no layer is specified, routing is performed on all layers.

`?generateSpine` *`g_generateSpine`*
: Specifies whether to call generate spine within Generate P2T Routing command.

`?spineLayer` *`g_spineLayer`*
: A specified target layer for generate spine and Generate P2T Routing commands.

`?connectPins` *`g_connectPins`*
: Specifies whether to start spine routing.

`?generateMesh` *`g_generateMesh`*
: Specifies whether to start the Generate Mesh Routing command.

`?meshBottomLayer` *`t_meshBottomLayer`*
: Specifies the starting layer of mesh routing.

`?meshTopLayer` *`t_meshTopLayer`*
: Specifies the ending layer of mesh routing.

`?meshSetting` *`l_meshSetting`*
: List of list for mesh routers sub-function setting on the corresponding layer.

`?meshExtendPreRoute` *`g_meshExtendPreRoute`*
: Specifies whether to extend the preroute trunks length to the specified region.

`?finishRouting` *`g_finishRouting`*
: Specifies whether to start the Finish routing command.

`?preferredLayerBottom` *`t_preferredLayerBottom`*
: Specifies the bottom layer of preferred routing layers for the Finish Routing command.

`?preferredLayerTop` *`t_preferredLayerTop`*

Specifies the top layer of preferred routing layers for the Finish Routing command.

`?postFixTrim` *`g_postFixTrim`*

Automatically adds trim shapes to routed nets to fix end-of-line (EOL) constraint violations.

`?postFixMinLength` *`g_postFixMinLength`*

Automatically extend routed pathseg and via shapes to satisfy various min length constraints, such as `minLength`, `minArea`, `minRectArea`, etcetera.

`?postFixEolKeepout` *`g_postFixEolKeepout`*

Whether or not to fix end-of-line keepout DRC violations after routing.

`?minSkipTrack` *`d_minSkipTrack`*

The minimum track number between meshing path segments on Mesh Count mode.

`?createFigGroup` *`g_createFigGroup`*

The name of the fig group to be created that contains all `oaFigs` included during this routing run. If set to `nil`, no fig group is created.

Value Returned

`t`	Nets were successfully routed without any errors.
`nil`	Routing was unsuccessful due to errors.

Examples

The following example runs Finish Routing with preferred bottom layer as `M1` and preferred top layer as `M4`.

```
vrtRouteAssisted(geGetEditCellView() `("net1") `((0.0 0.0) (1.0 1.0)) `("M1" "M2"
"M3" "M4" "M5" "M6") ?finishRouting t ?preferredLayerBottom "M1" ?preferredLayerTop
"M4")
```

In Assisted Routing mode, run Generate Mesh Routing with mesh bottom layer as `M1` and the mesh top layer as `M3`. Use `skipTrack` value of `1` for mesh layer `M1`, `meshCount` value of `5` for mesh layer `M2`, and `termAlign` value of `0` for mesh layer `M3`.

```
vrtRouteAssisted(geGetEditCellView() '("net1") '((0.0 0.0) (1.0 1.0)) '("M1" "M2"
"M3" "M4" "M5" "M6") ?generateMesh t ?meshBottomLayer "M1" ?meshTopLayer "M4"
?meshSetting ((nil skipTrack 1 layer "M1") (nil meshCount 5 layer "M2") (nil
termAlign 1 layer "M3") ) ?meshExtendPreRoute t)
```

In Assisted Routing mode, run Generate Spine with `M1` layer, and run Generate P2T Routing for pins to target `M1` spine layer with routable layers from `M1` to `M4`.

```
vrtRouteAssisted(geGetEditCellView() '("net1") '((0.0 0.0) (1.0 1.0)) '("M1" "M2"
"M3" "M4" "M5" "M6") ?generateSpine t ?spineLayer "M1" ?connectPins t
?preferredLayerBottom "M1" ?preferredLayerTop "M4" )
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

vrtRouteDesign

```
vrtRouteDesign(
     d_cvid
     [ layers l_layers ]
     [ ?nets l_nets ]
     [ ?regionBox l_regionBox ]
     [ ?reorderNets g_reorderNets ]
     [ ?excludeNets t_excludeNets ]
     [ ?constraintPriority g_constraintPriority ]
     [ ?preferredLayers l_preferredLayers ]
     [ ?viaCost f_viaCost ]
     [ ?connectBothGates g_connectBothGates ]
     [ ?preferConstraintViolationToOpen g_preferConstraintViolationToOpen ]
     [ ?postFixTrim g_postFixTrim ]
     [ ?postFixMinLength g_postFixMinLength ]
     [ ?postFixEolKeepout g_postFixEolKeepout ]
     [ ?enableDRDChecker g_enableDRDChecker ]
     [ ?appendDeviceLayers g_appenDeviceLayers ]
     [ ?deleteUnlockedShapes g_deleteUnlockedShapes ]
     [ ?logFile g_logFile ]
     [ ?appendToLog g_appendToLog
     [ ?createFigGroup g_createFigGroup ]
     )
     => t / nil
```

Description

Runs the device router on the selected design.

Arguments

d_cvid	Database ID of the cellview on which routing is to be run. The database ID can be obtained using the geGetEditCellView SKILL function. The default value is the cellview of the current window.
layerList *l_layerList*	The layers to be included for routing provided as a list of strings. By default, all device layers are included in this list, as specified by the `appendDeviceLayers` argument.
?nets *l_nets*	A list of net names to be routed. If no net is specified, it routes all the nets in the cellview.
?regionBox *l_regionBox*	The bounding box on which to run the router. All pins outside of this bounding box are ignored.

`?reordernets` *`g_reordernets`*

Whether or not to allow the routing algorithm to optimize the order in which nets are routed. If set to `nil`, the nets are routed in the order in which they are provided.

`?excludeNets` *`t_excludeNets`*

A string of net names not to be routed. This takes precedence over the nets parameter if the same net name is provided in both.

`?constraintPriority` *`g_constraintPriority`*

Considers each net's priority constraint when determining the order in which to route nets. If set to `nil`, nets are ordered without considering priority. If `reorderNets` is set to `nil`, setting `constraintPriority` has no effect because nets are not reordered. The default is `t`.

`?preferredLayers` *`l_preferredLayers`*

A list of layer names that the router preferably uses. The list must contain two elements, the lower layer provided before the higher layer. If set to `nil`, all layers have same preference and are routed.

`?viaCost` *`f_viaCost`*

The cost that the routing algorithm uses when considering creating a via. If the cost is `1`, each via's cost is equivalent to that of a pathseg with the same length and track width. If set to `nil`, the cost is determined by the routing algorithm.

`?connectBothGates` *`g_connectBothGates`*

Forces nets to connect to all the pins on the same poly gate, or to allow only one connection per gate.

`?preferConstraintViolationToOpen`
`g_preferConstraintViolationToOpen`

Determines whether the router should generate wires with constraint violations or instead leave them unrouted.

`?postFixTrim` *`g_postFixTrim`*

Automatically adds trim shapes to routed nets to fix end-of-line (EOL) constraint violations.

`?postFixMinLength` *`g_postFixMinLength`*

Automatically extend routed pathseg and via shapes to satisfy various min length constraints, such as `minLength`, `minArea`, `minRectArea`, etcetera.

`?postFixEolKeepout` *`g_postFixEolKeepout`*

Determines whether or not to fix end-of-line keepout DRC violations after routing.

`?enableDRDChecker` *`g_enableDRDChecker`*

Determines whether or not to generate constraint markers for all the nets that are included in this route. If `t`, the returned summary lists the DRCs field that contains the number of markers.

`?appendDeviceLayers` *`g_appenDeviceLayers`*

Indicates whether to automatically include all device layers in the design in the list of layers to route.

`?deleteUnlockedShapes` *`g_deleteUnlockedShapes`*

Determines whether or not to allow the router to improve the quality of solution by deleting any prerouted shapes that are not locked.

`?logFile` *`g_logFile`*

Specifies the name of a log file in which router messages are captured.

`?appendToLog` *`g_appendToLog`*

Specifies that new messages are appended to the log file if it already exists. If set to `nil`, the existing file is overwritten.

`?createFigGroup` *`g_createFigGroup`*

The name of the fig group to be created that contains all `oaFigs` included during this routing run. If set to `nil`, no fig group is created.

Value Returned

`t`	Returns a list containing several metrics about the routing run.
`nil`	Routing was unsuccessful due to errors.

Examples

The following example routes `net018` on layers `M1`, `M2`, `M3`, and `M4`, with layers `M3` and `M4` used preferentially. It also verifies that the number of opens is 0.

```
res = vrtRouteDesign(geGetEditCellView() '("M1" "M2" "M3" "M4") ?nets '("net018")
?preferredLayers '("M3" "M4"))
res
=> (("Time" "00:00:28")
    ("Opens" "0")
    ("Instances" "12")
    ("Shorts" "")
    ("Nets" "1")
    ("Wire Length" "10.97")
    ("DRCs" "Not Run")
    ("Vias" "30")
)
opensEntry = assoc("Opens" res)
=> ("Opens" "0")
numOpens = atoi(cadr(opensEntry))
=>  0
```

The following example route nets `net01` and `net02` under default settings. Any device layer is automatically included in the list of layers along with `M1`, `M2`, and `M3`.

```
vrtRouteDesign(geGetEditCellView() '("M1" "M2" "M3") ?nets("net01" "net02"))
```

The following example routes all of the nets in the design except for `net01` on the provided layers, considering only the pins in the bounding box defined by list(1:1 5:5). The order in which nets are routed is determined by the number of constraints, each gate's pins must be strongly connected, opens are preferred to constraint violations, and the design fixer attempts to fix trims and minLength violations. Although the device layer `Poly` is provided, any additional device layers are not included.

```
vrtRouteDesign(geGetEditCellView() '("Poly" "M1" "M2" "M3" "M4" "M5")
?nets nil
?excludeNets '("net01")
?reorderNets t
```

```
?regionBox list(1:1 5:5)
?viaCost 5
?prefferedLayers '("M3" "M4")
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

vrtCheckDesign

vrtStrandRoute

```
vrtStrandRoute(
     d_cvid
     [ layers l_layers ]
     [ spineLayer t_spineLayer ]
     [ createSpineSettings g_createSpineSettings ]
     [ routeSpineSettings g_routeSpineSettings ]
     [ ?nets l_nets ]
     [ ?region l_region ]
     [ ?preferredLayers l_preferredLayers ]
     [ ?viaCost f_viaCost ]
     [ ?connectBothGates g_connectBothGates ]
     [ ?preferVio2Open g_preferVio2Open ]
     [ ?postFixTrim g_postFixTrim ]
     [ ?postFixMinLength g_postFixMinLength ]
     [ ?postFixEolKeepout g_postFixEolKeepout ]
     )
     => t / nil
```

Description

Runs the ART spine generator and the spine router together.

Arguments

d_cvid	Database ID of the cellview. Can be obtained using the `geGetEditCellView` SKILL function.
`layers` *l_layers*	A list of routing layers.
`spineLayer` *t_spineLayer*	Specifies a spine layer.
`createSpineSettings` *g_createSpineSettings*	A DPL that specifies the settings for creating spines list. `list( nil 'enable nil|t)`
`routeSpineSettings` *g_routeSpineSettings*	

A DPL that specifies the settings for routing pins to spines.

```
list( nil 'enable nil|t
                'twigSharingDist nil|value
                'maxJogLength  nil|value
                'spinePinMaxDist nil|value
     )
```

`?nets` *`l_nets`*
: A list of net names to be routed. If no net is specified, it routes all the nets in the cellview.

`?region` *`l_region`*
: A list specifying the lower-left and upper-right coordinates of the region on which the function should operate.

`?preferredLayers` *`l_preferredLayers`*
: A list of preferred routing layers.

`?viaCost` *`f_viaCost`*
: The cost that the routing algorithm uses when considering creating a via. If the cost is `1`, each via's cost is equivalent to that of a pathseg with the same length and track width. If set to `nil`, the cost is determined by the routing algorithm.

`?connectBothGates` *`g_connectBothGates`*
: Forces nets to connect to all the pins on the same poly gate, or to allow only one connection per gate.

`?preferVio2Open` *`g_preferVio2Open`*
: Determines whether the router should generate wires with constraint violations or instead leave them unrouted. This is for the wires that cannot be routed without violation.

`?postFixTrim` *`g_postFixTrim`*
: Automatically adds trim shapes to routed nets to fix end-of-line (EOL) constraint violations.

`?postFixMinLength` *`g_postFixMinLength`*
: Automatically extend routed pathseg and via shapes to satisfy various min length constraints, such as `minLength`, `minArea`, `minRectArea`, etcetera.

`?postFixEolKeepout` *`g_postFixEolKeepout`*
: Determines whether or not to fix end-of-line keepout DRC violations after routing.

Value Returned

`t`	Spine generation and routing was successful.
`nil`	Spine generation and routing failed.

Examples

```
vrtStrandRoute( cvId list("MD" "PO" "M0" "M1" "M2" "M3") "M2"
            list( nil 'enable t)
            list( nil 'enable nil
                'twigSharingDist 0.72
                'maxJogLength 1.2
                'spinePinMaxDist nil )
            ?nets list("net21" "net34")
            ?preferredLayers list("M2" "M3")
)
```

Related Topics

Virtuoso Automated Placement and Routing SKILL Functions

20

Design Planning and Analysis Functions

This topic provides a list of Cadence® SKILL functions associated with the Virtuoso® Design Planning and Analysis flow.

Only the functions listed here are supported for public use. Any other functions, regardless of their name or prefix, and undocumented aspects of the functions described in this section, are private and subject to change at any time.

- lxDeleteVirtualPins
- lxGetVirtualFigGroupMasterName
- lxHiAdjustAreaBoundary
- lxHiAdjustBoundary
- lxHiCreateVirtGroup
- lxHiDesignPlanningOptions
- lxHiGenerateSelectedVirtualHierarchy
- lxHiGenerateVirtualHierarchy
- lxHiMakeCell
- lxHiMakeVirtualHierarchy
- lxHiRemaster
- lxHiSnapPatternOptions
- lxHiVirtHierOptions
- lxIsVirtualFigGroup
- lxMakeVirtualHierarchy
- lxVirtHierGetTermName
- lxVirtualHierarchyMakeCell

lxDeleteVirtualPins

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

Description

Deletes all virtual pins from selected virtual hierarchies.

Arguments

None

Value Returned

`t`	Virtual pins were deleted.
`nil`	Virtual pins could not be deleted.

Examples

```
lxDeleteVirtualPins()
```

Related Topics

Virtual Pin Snapping

lxGetVirtualFigGroupMasterName

```
lxGetVirtualFigGroupMasterName(
     d_figGroupID
     )
     => s_masterName
```

Description

Returns the schematic cellview name for the specified virtual figGroup in the format 'lib/cell/view'.

Arguments

d_figGroupID	ID of the virtual figGroup.

Value Returned

s_masterName	Master name of the specified virtual figGroup. If the specified figGroup is not a virtual figGroup or a created virtual figGroup, an empty string is returned.

Examples

```
lcv = dbOpenCellViewByName("XLME" "top" "layout")
fg = dbGetFigGroupByName(lcv "INV_0")
lxGetVirtualFigGroupMasterName(fg)
=> "XLME/INV/schematic"
>
```

Gets the schematic cellview name, `"XLME/INV/schematic"`, for the specified virtual figGroup.

Related Topics

Virtual Hierarchy Generation

lxHiAdjustAreaBoundary

```
lxHiAdjustAreaBoundary(
     )
     => t
```

Description

Opens the Adjust Boundary form, which can be used to modify the area boundary of the selected virtual figGroups.

Arguments

None

Value Returned

`t`	The Adjust Boundary form was opened.

Examples

```
lxHiAdjustAreaBoundary()
```

Related Topics

Adjust Boundary Form

lxHiAdjustBoundary

```
lxHiAdjustBoundary(
     )
     => t
```

Description

Adjusts the area boundary of the selected virtual figGroups or the PR Boundary of the selected soft blocks.

Arguments

None

Value Returned

`t`	The command was successful.

Examples

```
lxHiAdjustAreaBoundary()
```

Related Topics

Adjust Area Boundary Form

lxHiCreateVirtGroup

```
lxHiCreateVirtGroup(
     [ w_windowID ]
     )
     => t / nil
```

Description

Creates a virtual figGroup using the selected layout instances that are bound to schematic instances in the same design. When the `vhSelectionMode` environment variable is set to `t`, the SKILL function also creates virtual figGroups using selected layout devices that are bound to schematic instances from different designs. If a layout window is not specified, the function creates a virtual figGroup in the current window.

Arguments

w_windowID	ID of the window containing the layout instances for which the virtual figGroup needs to be created. Valid Values: Any window ID Default: The current window

Value Returned

`t`	The virtual figGroup was created.
`nil`	The virtual figGroup was not created.

Examples

```
lxHiCreateVirtGroup()
```

Creates a virtual figGroup for the selected layout instances in the current window.

Related Topics

vhSelectiveMode

Create Virtual Group Form

lxHiDesignPlanningOptions

```
lxHiDesignPlanningOptions(
     )
     => t
```

Description

Opens the Design Planning and Analysis Options form.

Arguments

None

Value Returned

`t`	The form was opened.
`nil`	The form was not opened.

Examples

```
lxHiDesignPlanningOptions()
```

Related Topics

Design Planning and Analysis Options Form

lxHiGenerateSelectedVirtualHierarchy

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

Description

Runs the *Generate Selected From Source* command with the *Design Planning – Virtual Hierarchy* option selected to generate virtual hierarchies for the selected schematic instances.

Arguments

None

Value Returned

`t`	The command was run.
`nil`	The command was not run.

Examples

```
lxHiGenerateSelectedVirtualHierarchy()
```

Related Topics

Generate Selected Components Form

lxHiGenerateVirtualHierarchy

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

Description

Runs the *Generate All From Source* command with the *Generate – Design Planning – Virtual Hierarchy* option selected to generate virtual hierarchies.

Arguments

None

Value Returned

`t`	The command was run.
`nil`	The command was not run.

Examples

```
lxHiGenerateVirtualHierarchy()
```

Related Topics

Generate Layout Form

lxHiMakeCell

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

Description

Opens the Design Planner Make Cell form, which can be used to create a new cellview using the selected virtual hierarchy from the top-cell layout.

Arguments

None

Value Returned

`t`	The Make Cell form was opened.
`nil`	The Make Cell form was not opened.

Examples

```
lxHiMakeCell()
```

Related Topics

Make Cell Form

lxHiMakeVirtualHierarchy

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

Description

Opens the Make Virtual Hierarchy form, which can be used to create a virtual hierarchy at the top-cell level using the selected layout instance.

Arguments

None

Value Returned

`t`	The Make Virtual Hierarchy form was opened.
`nil`	The Make Virtual Hierarchy form was not opened.

Examples

```
lxHiMakeVirtualHierarchy()
```

Related Topics

Make Virtual Hierarchy Form

lxHiRemaster

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

Description

Opens the Remaster form, which can be used to replace the selected virtual hierarchy with an existing layout variant.

Arguments

None

Value Returned

`t`	The Remaster form was opened.
`nil`	The Remaster form was not opened.

Examples

```
lxHiRemaster()
```

Related Topics

Remaster Form

lxHiSnapPatternOptions

```
lxHiSnapPatternOptions(
     )
     => t
```

Description

Opens the Design Planning and Analysis Options form to create snap pattern grids and specify that horizontal and vertical tracks be created inside the PR boundary.

Arguments

None

Value Returned

`t`	The form was opened.
`nil`	The form was not opened.

Examples

```
lxHiSnapPatternOptions()
```

Related Topics

Design Planning and Analysis Options Form

lxHiVirtHierOptions

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

Description

Opens the Design Planning and Analysis Options form, which can be used to specify whether the layout displays the name of the virtual hierarchy, the cell, or both. In addition, the form can be used to enable or disable symbol overlay and to specify whether the virtual hierarchy can be auto placed during editing.

Arguments

None

Value Returned

`t`	The Design Planning and Analysis Options form was opened.
`nil`	The Design Planning and Analysis Options form was not opened.

Examples

```
lxHiVirtHierOptions()
```

Related Topics

Design Planning and Analysis Options Form

lxIsVirtualFigGroup

```
lxIsVirtualFigGroup(
     d_figID
     )
     => t / nil
```

Description

Checks if the specified figGroup is a virtual figGroup.

Arguments

None

Value Returned

`t`	The specified figGroup is a virtual figGroup.
`nil`	The specified figGroup is not a virtual figGroup.

Examples

```
lxIsVirtualFigGroup()
```

Related Topics

Virtual Hierarchy Generation

lxMakeVirtualHierarchy

```
lxMakeVirtualHierarchy(
     d_layCellViewID
     l_insts
     [ ?allInstsSameMaster { t | nil } ]
     [ ?preserveVirtualPins { t | nil } ]
     [ ?keepLabels { t | nil } ]
     [ ?placementStatus { Fixed | Locked | None } ]
     )
     => t / nil
```

Description

Creates virtual hierarchies from the cells of the listed bound instances.

Arguments

d_layCellViewID	Database ID of the layout cellview containing the instances to be replaced with virtual hierarchies.
l_insts	Database IDs of the instances to be converted to virtual hierarchies.
?allInstsSameMaster	Controls whether all the instances of the selected layout master are replaced with the same virtual hierarchy. Environment variable: makeVirtualAllInstsSameMaster
?preserveVirtualPins	Controls whether pinFigs from cell masters are retained as virtual pin shapes in the virtual hierarchies, to be used later to create pin shapes in the made cellview. Environment variable: makeVirtualPreserveVirtualPins
?keepLabels	Controls whether labels on shapes are preserved on the made virtual hierarchy. Environment variable: flattenKeepLabels
?placementStatus	Specifies the placement status of the virtual hierarchies created by the command. The valid values are: `"Fixed"`, `"Locked"`, and `"None"`.

Value Returned

t	Virtual hierarchies are created.
nil	The command failed.

Examples

Create virtual hierarchies for instances “I1” “I2” “I3” “I4”, but not all instances of their masters, preserving pins as virtual pins with no labels.

```
lcv = dbOpenCellViewByType("libA" "top" "layout" "maskLayout" "a")
i1 = dbFindAnyInstByName(lcv "I1")
i2 = dbFindAnyInstByName(lcv "I2")
i3 = dbFindAnyInstByName(lcv "I3")
```

```
i4 = dbFindAnyInstByName(lcv "I4")
lxMakeVirtualHierarchy(lcv
list(i1 i2 i3 i4) ?allInstsSameMaster nil ?preserveVirtualPins t ?keepLabels nil)
```

Related Topics

Make Virtual Hierarchy Command

lxVirtHierGetTermName

```
lxVirtHierGetTermName(
     d_shapeID
     )
     => s_termName / nil
```

Description

Returns the associated terminal name if the selected shape is a virtual pin.

Arguments

d_shapeID	Database ID of the virtual pin shape.

Value Returned

s_termName	Terminal name associated with the selected virtual pin shape.
nil	The command failed.

Examples

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
     figGroup = dbGetFigGroupByName(cellView "I1")
     foreach(fig figGroup~>figs
       virtualPinTermName = lxVirtHierGetTermName(fig)
       .
       .
       .
     )
  )
)
>
```

Related Topics

Creating Virtual Pins

lxVirtualHierarchyMakeCell

```
lxVirtualHierarchyMakeCell(
     d_layCellViewID
     l_virtHiers
     [ ?lib { libName | nil } ]
     [ ?cell { cellName | nil } ]
     [ ?view { viewName | nil } ]
     [ ?cellType { softMacro | none | digital softMacro | block } ]
     [ ?overWriteCV { t | nil } ]
     [ ?allClones { t | nil } ]
     [ ?allLevels { t | nil } ]
     [ ?pinsChoice { Congestion aware | On boundary | Below boundary | Promote pins
     } ]
     [ ?deleteVirtualPins { t | nil } ]
     [ ?pushIntoBlock { t | nil } ]
     [ ?pushRoutesAsBlockages { t | nil } ]
     [ ?pushInternalRoutesOnly { t | nil } ]
     )
     => l_layCellViewID / nil
```

Description

Creates real cellviews for the specified virtual hierarchies.

Arguments

d_layCellViewID	Database ID of the layout cellview containing the virtual hierarchies to be replaced with real cellviews.
l_virtHiers	Database IDs of the virtual hierarchy figGroups to be made into real cellviews.
?lib	Name of the library to be used for creating a cellview for a virtual hierarchy. When the allLevels argument is enabled or multiple virtual hierarchies need to be created, the library name to use is determined by the matching schematic hierarchy for each virtual hierarchy. This is also the case if library name is not specified.
?cell	Name of the cell to be used for creating a cellview for a virtual hierarchy. When the allLevels argument is enabled or multiple virtual hierarchies need to be created, the cell name to use is determined by the matching schematic hierarchy for each virtual hierarchy. This is also the case if library name is not specified.
?view	Name of the view to be used for creating a cellview for a virtual hierarchy. If a view name is not specified, the default view name, layout_variant_1 is used. The subsequent view names are then automatically incremented to layout_variant_2, and so on.

`?cellType`

Specifies the type of cell to be created for the selected virtual hierarchy.

- `softMacro` creates a soft block type cellview.
- `none` creates a custom cell type.
- `digital softMacro` creates a soft block with hierarchy for a block of type `digital`.
- `block` creates hard macros that can be supported by macro placers, Virtuoso Layout Suite XL commands such as *Load Physical View*, and other applications that support only macros and blocks.

Environment variable: makeCellType

`?overwriteCV`

Overwrites an existing cellview when creating the new one.

Environment variable: makeCellOverwriteLayout

`?allClones`

Replaces all the clones of the selected virtual hierarchy with the new cellview.

Environment variable: makeCellVirtualClones

`?allLevels`

Replaces the selected virtual hierarchy and all the virtual hierarchy levels inside the selected virtual hierarchy with a new cellview.

Environment variable: makeCellAllLevels

`?pinsChoice`

Specifies the options for creating interface pins on the made cellview for the selected virtual hierarchy.

- `Congestion aware` runs the congestion-aware global router to automatically create pins on the boundary of the virtual hierarchy.
- `On boundary` creates pins on the boundary of the virtual hierarchy, ensuring the shortest possible net length in the direction of routing.
- `Below boundary` creates pins just below the boundary of the virtual hierarchy.
- `Promote pins` extends pins from lower levels of a virtual hierarchy to a higher level.

Environment variable: makeCellPinsChoice

`?deleteVirtualPins`	Deletes any existing virtual pins in the specified virtual hierarchies. Environment variable: makeCellDeleteVirtualPins
`?pushIntoBlock`	Pushes overlapping routes, blockages, rows, and width spacing patterns into the made cellview. Environment variable: makeCellPushInBlock
`?pushRoutesAsBlockages`	Pushes overlapping routes as blockages into the made cellview. This argument can be specified only when `?pushIntoBlock` is enabled. Environment variable: makeCellPushRoutesAsBlockages
`?pushInternalRoutesOnly`	Pushes only the internal routes into the made cellview. This argument can be specified only when `?pushIntoBlock` is enabled. Environment variable: makeCellPushInternalRoutesOnly

Value Returned

`l_layCellViewID`	List of layout cellview IDs created by the command.
`nil`	The command failed.

Examples

Creates a cellview for virtual hierarchy “`I1`”. The lib cellview should be “`libB`” “`cell1`” “`layout`”, the type should be “`softMacro`”, and pin creation should be below boundary.

```
lcv = dbOpenCellViewByType("libA" "top" "layout" "maskLayout" "a")
vh1 = dbGetFigGroupByName(lcv "I1")
lxVirtualHierarchyMakeCell(lcv1 list(vh1) ?lib "libB" ?cell "cell1" ?view "layout"
?cellType "softMacro" ?allLevels nil ?pinsChoice "Below boundary")
```

Related Topics

Make Cell Command

21

Virtuoso Concurrent Layout Functions

This topic provides a list of Cadence® SKILL functions associated with Virtuoso Concurrent Layout.

Only the functions listed here are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

Concurrent Layout Interface Functions

Use the functions described in this section to get the original library cell view information, the concurrent layout editor mode in which the cellview is open, or switch the current mode.

- cliGetCellViewOrigCellName
- cliGetCellViewOrigLibName
- cliGetCellViewOrigViewName
- cliIsCVType
- cliIsDesignerMode
- cliIsManagerMode
- cliReopen
- cliSave

cliGetCellViewOrigCellName

```
cliGetCellViewOrigCellName(
     d_cellViewID
     )
     => t_cellName / nil
```

Description

Returns the cell name of the specified Concurrent Layout cellview.

Arguments

d_cellviewID	Database ID of a cellview.

Value Returned

t_cellName	Name of the cell containing the cellview.
nil	Fails to call the function.

Examples

Returns the cell name of the opened `test/demo/layout_cle_p1` cellview.

```
cleWin = deOpenCellView("test" "demo" "layout_cle_p1" "maskLayout" list(list(0 0)
list(500 600)) "a")
cliGetCellViewOrigCellName(geGetEditCellView(cleWin))
=> "demo"
```

cliGetCellViewOrigLibName

```
cliGetCellViewOrigLibName(
     d_cellViewID
     )
     => t_libName / nil
```

Description

Returns the library name of the specified Concurrent Layout cellview.

Arguments

d_cellviewID	Database ID of a cellview.

Value Returned

t_libName	Name of the library containing the cellview.
nil	Fails to call the function.

Examples

Returns the library name of the opened `test/demo/layout_cle_p1` cellview.

```
cleWin = deOpenCellView("test" "demo" "layout_cle_p1" "maskLayout" list(list(0 0)
list(500 600)) "a")
cliGetCellViewOrigLibName(geGetEditCellView(cleWin))
=> "test"
```

cliGetCellViewOrigViewName

```
cliGetCellViewOrigViewName(
     d_cellViewID
     )
     => t_viewName / nil
```

Description

Returns the view name of specified Concurrent Layout cellview.

Arguments

d_cellviewID	Database ID of a cellview.

Value Returned

t_viewName	Name of the view containing the cellview.
nil	Fails to call the function.

Examples

Returns the view name of the opened `test/demo/layout_cle_p1` cellview.

```
cleWin = deOpenCellView("test" "demo" "layout_cle_p1" "maskLayout" list(list(0 0)
list(500 600)) "a")
cliGetCellViewOrigViewName(geGetEditCellView(cleWin))
=> "layout_cle_p1"
```

cliIsCVType

```
cliIsCVType(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Checks whether the cellview specified by library, cell, and view is a Concurrent Layout cellview. A Concurrent Layout cellview can be a top cellview initialized for Concurrent Layout editing, a Concurrent Layout design partition view, or a Concurrent Layout scratch cellview.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.

Value Returned

t	The specified cellview is a Concurrent Layout cellview.
nil	The specified cellview is not related to Concurrent Layout.

Examples

Confirms that the cellview with library `myLib`, cell `myCell`, and view `myView` is a Concurrent Layout cellview.

```
cliIsCVType("myLib" "myCell" "myView")
=>t
```

cliIsDesignerMode

```
cliIsDesignerMode(
     d_cellViewID
     )
     => t / nil
```

Description

Checks whether the specified cellview is open in Concurrent Layout designer mode.

Arguments

d_cellviewID	Database ID of a cellview.

Value Returned

`t`	The specified cellview is open in designer mode.
`nil`	The specified cellview is not open in designer mode.

Examples

Confirms that the cellview `cvId` is open in designer mode.

```
cliIsDesignerMode(cvId)
=>t
```

cliIsManagerMode

```
cliIsManagerMode(
     d_cellViewID
     )
     => t / nil
```

Description

Checks whether the specified cellview is open in Concurrent Layout manager mode.

Arguments

d_cellViewID	Database ID of a cellview.

Value Returned

`t`	The specified cellview is open in manager mode.
`nil`	The specified cellview is not open in manager mode.

Examples

Confirms that the cellview `cvId` is open in manager mode.

```
cliIsManagerMode(cvId)
=> t
```

cliReopen

```
cliReopen(
     d_cellViewID
     t_mode
     )
     => t / nil
```

Description

Changes the mode of the specified cellview to the given mode. This function is equivalent to the `dbReopen` function but can handle both Concurrent Layout and non-Concurrent Layout cellviews.

Arguments

d_cellViewID	Database ID of a cellview.
t_mode	A string representing the new mode. Valid values: ■ `r` - Reopens the cellview in read mode. Changes are discarded when you switch from edit to read mode. ■ `a` - Reopens the cellview in append mode. The cellview might refresh when you switch from read or scratch mode to append mode. For example, the cellview data is refreshed only if there are timestamp changes on the disk file. ■ `w` - Reopens the cellview in write mode. In this mode, the contents of the cellview are truncated. ■ `s` - Reopens the cellview in scratch mode.

Value Returned

`t`	The mode change is successful.
`nil`	The mode change is not successful.

Examples

Opens the cellview `cvId` in append mode:

```
cliReopen(cvId "a")
=> t
```

Related Topics

`dbReopen`

cliSave

```
cliSave(
     d_cellViewID
     )
     => t / nil
```

Description

Saves the changes made to the specified cellview. This function is equivalent to the `dbSave` function but can handle both Concurrent Layout and non-Concurrent Layout cellviews.

Arguments

d_cellViewID	Database ID of the cellview to be saved.

Value Returned

`t`	The specified cellview is saved.
`nil`	The specified cellview is not saved.

Examples

Saves the cellview `cvId`.

```
cliSave(cvId)
=> t
```

Related Topics

dbSave

Concurrent Layout Editing Functions

Use the functions described in this section to perform edit operations such as create partitions, auto save files, and clean scratch cellviews.

- cleCleanUpScratchCellViews
- cleCreatePartition
- cleCreatePartitionView
- cleGetCurrentPartition
- cleGetPartitions
- cleHasAutoSavedFile
- cleHasPanicFile
- cleIsAreaBasedPartition
- cleIsLayerBasedPartition
- cleOpenAutoSavedCellView
- cleOpenPanicCellView
- clePartitionAttachAreaBoundary
- clePartitionGetAreaBoundaries
- clePartitionGetLayers
- clePartitionGetStatus
- cleRefreshAssistantByCellView
- cleReinitialize
- cleRestoreAndOpenAutoSavedFile
- cleRestoreAndOpenPanicFile
- cleStretchMalformedSpine

cleCleanUpScratchCellViews

```
cleCleanUpScratchCellViews(
     t_libName
     [ ?discardPanic g_value ]
     [ ?discardAutoSaved g_value ]
     )
     => t  / nil
```

Description

Deletes all the scratch cellviews with the prefix `zcleSCRATCH`. These cellviews are saved by the auto-saved or panic files.

Arguments

t_libName	Name of a library.
?discardPanic *g_value*	Discards the scratch cellview from the saved panic file. The default is `nil`.
?discardAutoSaved *g_value*	Discards the scratch cellview from the auto-saved file. The default is `nil`.

Value Returned

`t`	Scratch cellviews are deleted.
`nil`	Scratch cellviews are not deleted or, there are no scratch cellviews to be deleted.

Examples

Checks if a panic file is present in the Concurrent Layout design partition view `CustomDigital/cleanUp/layout_cle_p1`.

```
cleHasPanicFile("CustomDigital" "cleanUp" "layout_cle_p1")
=> t
```

Removes the panic scratch cellview.

```
cleCleanUpScratchCellViews("CustomDigital" ?discardPanic t)
```

```
(CLE-101062): Deleted 1 unused scratch cellviews from the disk.
 - CustomDigital/zcleSCRATCH_cleanUp_<userId>_userEdit_panic/layout_cle_p1
=> t
```

Checks if an auto-saved file is present in the Concurrent Layout design partition view `CustomDigital/cleanUp/layout_cle_p1`.

```
cleHasAutoSavedFile("CustomDigital" "cleanUp" "layout_cle_p1")
=> t
```

Deletes the auto-saved scratch cellview.

```
cleCleanUpScratchCellViews("CustomDigital" ?discardAutoSaved t)
(CLE-101062): Deleted 1 unused scratch cellviews from the disk.
 - CustomDigital/zcleSCRATCH_cleanUp_<userId>_userEdit_auto/layout_cle_p1
=> t
```

cleCreatePartition

```
cleCreatePartition(
     d_cellViewID
     t_name
     [ t_owner ]
     )
     => t_designPartitionName / nil
```

Description

Creates a design partition definition in the specified Concurrent Layout cellview and changes the status of the design partition to *Defined*.

Arguments

d_cellViewID	Database ID of a cellview.
t_name	Name of the design partition. If you specify an empty string, a unique name is generated automatically. The syntax for this name is `cle_px`, where *x* is a number.
t_owner	Owner of the design partition.

Value Returned

t_designPartitionName	Name of the design partition that is created.
`nil`	Design partition is not created.

Examples

Creates the design partition `cle_p1` in the cellview `cvId`.

```
cleCreatePartition(cvId "cle_p1")
=> "cle_p1"
```

Creates the design partition `cle_p2` with the specified owner `"James"` in the cellview `cvId`.

```
cleCreatePartition(cvId "cle_p2" "James")
=> "cle_p2"
```

cleCreatePartitionView

```
cleCreatePartitionView(
     d_cellViewID
     t_designPartition
     )
     => t / nil
```

Description

Creates a design partition view on disk for the specified design partition in the specified Concurrent Layout cellview and changes the status of the design partition to *Created*.

Arguments

d_cellViewID	Database ID of a cellview.
t_designPartition	Name of the design partition.

Value Returned

`t`	Design partition view is created.
`nil`	Design partition view is not created.

Examples

Creates the design partition view on disk for the design partition `cle_p1` in the cellview `cvId`.

```
cleCreatePartitionView(cvId "cle_p1")
=> t
```

cleGetCurrentPartition

```
cleGetCurrentPartition(
     d_cellViewID
     )
     => t_partitionName / nil
```

Description

Retrieves the name of the Concurrent Layout design partition in which you are currently editing.

Arguments

d_cellviewID	Database ID of the cellview from which you want to retrieve the name of the design partition in which you are currently editing.

Value Returned

t_partitionName	Name of the design partition in which you are currently editing.
nil	You are not editing in a design partition.

Examples

Indicates that the cellview `cvId` is open in the Concurrent Layout designer mode and you are currently editing in the design partition `cle_p1`.

```
cleGetCurrentPartition(cvId)
=> "cle_p1"
```

cleGetPartitions

```
cleGetPartitions(
     d_cellViewID
     )
     => l_partitionNames / nil
```

Description

Lists the names of all Concurrent Layout design partitions defined in the specified cellview.

Arguments

d_cellViewID	Database ID of the cellview from which you want to retrieve the design partition names.

Value Returned

l_partitionNames	List of names of all design partitions defined in the specified cellview.
nil	There are no design partitions defined in the specified cellview.

Examples

Lists the names of three design partitions `cle_p1`, `cle_p2`, and `cle_p3` defined in the cellview `cvId`.

```
cleGetPartitions(cvId)
=> ("cle_p1" "cle_p2" "cle_p3")
```

cleHasAutoSavedFile

```
cleHasAutoSavedFile(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Checks whether the specified Concurrent Layout cellview has an auto-saved file.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.

Value Returned

t	The auto-saved file is present in the specified Concurrent Layout cellview.
nil	The auto-saved file is not present in the specified Concurrent Layout cellview.

Examples

Checks if the auto-saved file is present in the Concurrent Layout design partition view `designLib/testing1/layout_cle_p1`.

```
cleHasAutoSavedFile("designLib" "testing1" "layout_cle_p1")
=> t
```

cleHasPanicFile

```
cleHasPanicFile(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Checks whether the specified Concurrent Layout cellview has a panic file.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.

Value Returned

t	Panic file is present in the specified Concurrent Layout cellview.
nil	Panic file is not present in the specified Concurrent Layout cellview.

Examples

Checks if a panic file is present in the Concurrent Layout design partition view `designLib/testing1/layout_cle_p1`.

```
cleHasPanicFile("designLib" "testing1" "layout_cle_p1")
=> t
```

cleIsAreaBasedPartition

```
cleIsAreaBasedPartition(
     d_cellViewID
     t_name
     )
     => t / nil
```

Description

Checks whether the specified design partition is defined by area boundaries.

Arguments

d_cellViewID	Database ID of a cellview.
t_name	Name of the design partition.

Value Returned

t	The specified design partition is an area-based design partition.
nil	The specified design partition is not an area-based design partition.

Examples

Confirms that design partition `cle_p1` in the cellview `cv` is an area-based design partition.

```
cleIsAreaBasedPartition(cv "cle_p1")
=> t
```

cleIsLayerBasedPartition

```
cleIsLayerBasedPartition(
     d_cellViewID
     t_name
     )
     => t / nil
```

Description

Checks whether the specified Concurrent Layout design partition is defined by a layer range.

Arguments

d_cellViewID	Database ID of the cellview in which the design partition you want to check is defined.
t_name	Name of a design partition.

Value Returned

`t`	The specified design partition is a layer-based design partition.
`nil`	The specified design partition is not a layer-based design partition.

Examples

Confirms that the design partition `cle_p1` is a layer-based design partition.

```
cleIsLayerBasedPartition(cvId "cle_p1")
=> t
```

cleOpenAutoSavedCellView

```
cleOpenAutoSavedCellView(
     t_libName
     t_cellName
     t_viewName
     [ g_openWindow ]
     )
     => w_windowID / d_cellviewID / nil
```

Description

Opens the auto-saved cellview in the read-only mode. You can save the data using the Save a Copy command in the *File* menu of the layout window to save the cellview back to the disk. The auto-saved file is the cellview file name appended by a plus sign.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.
g_openWindow	Specifies whether to open a layout window. Default value is `t`.

Value returned

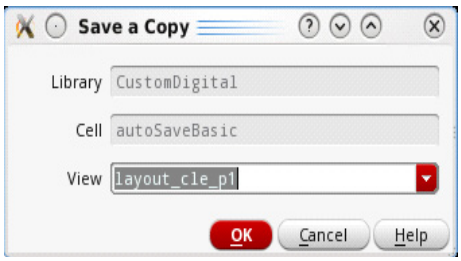
w_windowID	Window ID of the Concurrent Layout cellview that opens when *g_openWindow* is set to `t`.
d_cellviewID	Database ID of the Concurrent Layout cellview that opens if *g_openWindow* is set to `nil`.
`nil`	An error occurred or file does not exists.

Examples

If the auto-saved file exists in the `layout_cle_p1` view of `autoSaveBasic` in the `CustomDigital` library, open the canvas window and save it on the disk.

```
cleHasAutoSavedFile("CustomDigital" "autoSaveBasic" "layout_cle_p1")
=> t
cleOpenAutoSavedCellView("CustomDigital" "autoSaveBasic" "layout_cle_p1")
```

Select the *Save a Copy* command from the *File* menu, then click the *OK* button.



cleOpenPanicCellView

```
cleOpenPanicCellView(
     t_libName
     t_cellName
     t_viewName
     [ g_openWindow ]
     )
     => w_windowID / d_cellviewID / nil
```

Description

Opens a saved panic Concurrent Layout cellview from a physical panic file with the `.oa-` extension. This cellview opens in the read-only mode. You can save the data using the *Save a Copy* command in the *File* menu to a new cellview or overwrite the original Concurrent Layout cellview on the disk.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.
g_openWindow	Specifies whether to open a layout window. Default value is `t`.

Value Returned

w_windowID	Window ID of the Concurrent Layout cellview that opens when *g_openWindow* is set to `t`.
d_cellviewID	The dbobject of the Concurrent Layout cellview that opens if *g_openWindow* is set to `nil`.
`nil`	An error has occurred.

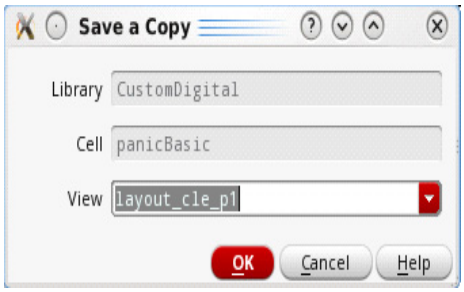
Examples

If a panic cellview exists in the `layout_cle_p1` view of `panicBasic` in the library `CustomDigital`, open the canvas window and then use the *Save as Copy* command to copy the result to the original cellview.

```
cleHasPanicFile("CustomDigital" "panicBasic" "layout_cle_p1")
=> t
```

```
cleOpenPanicCellView("CustomDigital" "panicBasic" "layout_cle_p1")
```

Select the *Save a Copy* command from the *File* menu, then click the *OK* button.



clePartitionAttachAreaBoundary

```
clePartitionAttachAreaBoundary(
     d_cellViewID
     t_designPartition
     { d_areaBoundaryID| t_areaBoundaryName }
     )
     => t / nil
```

Description

Attaches the name or ID of an existing area boundary to the specified design partition in the specified Concurrent Layout cellview.

Arguments

d_cellViewID	Database ID of a cellview.
t_designPartition	Name of the design partition.
d_areaBoundaryID | *t_areaBoundaryName*	Database ID or the name of the area boundary.

Value Returned

t	Area boundary is attached to the specified design partition.
nil	Area boundary is not attached to the specifeid design partition.

Examples

Attaches the existing area boundary `AB_1` to the design partition `cle_p1` in the cellview `cvId`.

```
clePartitionAttachAreaBoundary(cvId "cle_p1" "AB_1")
=> t
```

Finds the database ID of an existing area boundary `AB_1` and then attaches it to the design partition `cle_p1` in the cellview `cvId`.

```
areaID = dbFindAreaBoundaryByName(cvId "AB_1")
clePartitionAttachAreaBoundary(cvId "cle_p1" areaID)

=> t
```

clePartitionGetAreaBoundaries

```
clePartitionGetAreaBoundaries(
     d_cellViewID
     t_name
     )
     => l_areaBoundaries / nil
```

Description

Returns a list of database IDs of area boundaries attached to the specified design partition.

Arguments

d_cellViewID	Database ID of the cellview in which the design partition is defined.
t_name	Name of the design partition.

Value Returned

l_areaBoundaries	List of database IDs of area boundaries associated with the specified design partition.
nil	No area boundary is associated with the specified design partition.

Examples

Returns the list of database IDs of area boundaries associated with the design partition `cle_p1`.

```
clePartitionGetAreaBoundaries(geGetEditCellView() "cle_p1")
=> (db:0x2200c59a db:0x2200c59c)
```

Returns the list of the names of area boundaries associated with the design partition `cle_p2`.

```
clePartitionGetAreaBoundaries(geGetEditCellView() "cle_p2")~>name
=> ("P2_AB1 "P2_AB2")
```

clePartitionGetLayers

```
clePartitionGetLayers(
     d_cellViewID
     t_name
     )
     => l_layers / nil
```

Description

Retrieves the list of layers included in the specified Concurrent Layout design partition.

Arguments

d_cellviewID	Database ID of the cellview in which the design partition you want to check is defined.
t_name	Name of the design partition from which you want to retrieve the layer range.

Value Returned

l_layers	List of layer numbers included in the specific design partition. Layer numbers are defined in the technology database attached to the cellview.
`nil`	No layers are included in the specified design partition.

Examples

Indicates that layer numbers `30`, `32`, `34`, `36`, and `38` are included in design partition `cle_p1`.

```
layers = clePartitionGetLayers(cvId "cle_p1")
=> (30 32 34 36 38)
```

clePartitionGetStatus

```
clePartitionGetStatus(
     d_cellviewID
     t_name
     )
     => l_partitionInfo / nil
```

Description

Retrieves the status and additional information of the specified Concurrent Layout Editing design partition.

Arguments

d_cellViewID	Database ID of a cellview.
t_name	Name of a design partition.

Value Returned

l_partitionInfo	Returns a list with the `status` information of the specified design partitions, where, The `status` is the design partition view state. Valid values for the status are: ■ `Defined`: A design partition is added to the cellview and the corresponding design partition view is not created. ■ `Created`: A design partition is added to the cellview and the corresponding partition view is created. ■ `Submitted`: Changes made in the design partition view for specified design partition have been submitted for merge. ■ `Not Submitted`: Changes made in the design partition view for the specified design partition have not been submitted for merge. ■ `Rejected`: Changes made in the design partition view of the specified design partition are rejected by the manager.

- `Merged`: Changes made in the design partition view of the specified design partition were merged by the manager and the merged result was not committed.
- `Committed`: Changes made in the design partition view of the specified design partition were merged by manager and the merged result was committed.
- `Reuse`: A design partition view without a corresponding design partition can be assigned to other design partitions.
- `Reset`: Changes in the design partition view of the specified design partition are reset by the designer after the changes are merged and committed.
- `Error`: Specifies that the design partition view does not belong to the current Concurrent Layout top design.

You can query this using the following command:

```
l_partitionInfo->status
```

`nil` Design partition does not exist.

Examples

Shows that the design partition `cle_p1` is added to the cellview, but is not created on disk.

```
clePartitionGetStatus(cv "cle_p1")
=> (nil status "Defined")
```

Shows that the design partition view `cle_p1` is created for the specified design partition `cle_p1`.

```
clePartitionGetStatus(cv "cle_p1")->status
=> "Created"
```

Shows that changes in the design partition view of the design partition `cle_p1` are saved.

```
clePartitionGetStatus(cv "cle_p1")->status
=> "Not Submitted"
```

cleRefreshAssistantByCellView

```
cleRefreshAssistantByCellView(
     d_cellViewID
     )
     => t / nil
```

Description

Refreshes the content of the Concurrent Layout assistant in the specified Concurrent Layout cellview when you load a SKILL script that updates the design partition definition or changes the status of design partition view.

Arguments

d_cellViewID	Database ID of a cellview.

Value Returned

t	The assistant information is refreshed.
nil	The assistant information is not refreshed.

Examples

Creates the design partition `cle_p1` in the cellview `cvId` and changes the status of `cle_p1` to *Defined*.

```
cleCreatePartition(cvId "cle_p1")
=> "cle_p1"
```

Attaches existing area boundary `AB_1` to `cle_p1`.

```
clePartitionAttachAreaBoundary(cvId "cle_p1" "AB_1")
=> t
```

Creates the design partition view corresponding to the design partition `cle_p1` and changes the status of `cle_p1` to *Created*.

```
cleCreatePartitionView(cvId "cle_p1")
=> t
```

Creates design partition `cle_p2` in the cellview `cvId` and changes the status of `cle_p2` to *Defined*.

```
cleCreatePartition(cvId "cle_p2")
=> "cle_p2"
```

Creates the design partition view corresponding to the design partition `cle_p2` and changes the status of `cle_p2` to *Created*.

```
cleCreatePartitionView(cvId "cle_p2")
=> t
```

Refreshes the Concurrent Layout assistant to reflect the changes made in the preceding steps.

```
cleRefreshAssistantByCellView(cvId)
=> t
```

cleReinitialize

```
cleReinitialize(
     w_windowID
     )
     => t / nil
```

Description

Reinitializes a Concurrent Layout cellview if there are objects added to it outside the Concurrent Layout environment.

Note: Save the reinitialized cellview before you start editing it in concurrent layout.

Arguments

w_windowID	Window ID of the edit cellview you want to reinitialize.

Value Returned

`t`	The edit cellview of the specified window has been reinitialized.
`nil`	The edit cellview of the specified window is not reinitialized.

Examples

The edit cellview open in the window `winId` is reinitialized for Concurrent Layout editing.

```
cleReInitialize(winId)
=> t
```

cleRestoreAndOpenAutoSavedFile

```
cleRestoreAndOpenAutoSavedFile(
     t_libName
     t_cellName
     t_viewName
     [ g_openWindow ]
     )
     => w_windowID / d_cellviewID / nil
```

Description

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

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.
g_openWindow	Specifies whether to open a layout window. Default value is `t`.

Value Returned

w_windowID	Window ID of the Concurrent Layout cellview that opens when *g_openWindow* is set to `t`.
d_cellviewID	Database ID of the Concurrent Layout cellview that opens if *g_openWindow* is set to `nil`.
`nil`	An error has occurred or file does not exist.

Examples

Checks if the cellview has a auto-saved file.

```
cleHasAutoSavedFile("designLib" "testing1" "layout_cle_p1")
=> t
```

Restores and open the auto-saved file.

```
autoSavedCV = cleRestoreAndOpenAutoSavedFile("designLib" "testing1"
"layout_cle_p1")
```

cleRestoreAndOpenPanicFile

```
cleRestoreAndOpenPanicFile(
     t_libName
     t_cellName
     t_viewName
     [ g_openWindow ]
     )
     => w_windowID / d_cellviewID / nil
```

Description

Restores and opens the panic file for the specified Concurrent Layout cellview.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.
g_openWindow	Specifies whether to open a layout window. Default value is `t`.

Value Returned

w_windowID	Window ID of the Concurrent Layout cellview that opens when *g_openWindow* is set to `t`.
d_cellviewID	Database ID of the Concurrent Layout cellview that opens if *g_openWindow* is set to `nil`.
`nil`	An error has occurred or file does not exist.

Examples

Checks if the cellview `testing1` has a panic file.

```
cleHasPanicFile("designLib" "testing1" "layout_cle_p1")
=> t
```

Restores and opens the panic file.

```
panicCV = cleRestoreAndOpenPanicFile("designLib" "testing1" "layout_cle_p1")
```

cleStretchMalformedSpine

```
cleStretchMalformedSpine(
     g_enable
     )
     => t / nil
```

Description

Specifies whether the *Stretch* command can recognize a malformed spine and move it together if possible. For example, a malformed spine is a spine without a turn containing several collinear pathSegs with non-coincident end points.

By default, this is disabled in a Concurrent Layout design (manager or designer mode) so that stretching a split pathSeg inside a design partition will not attempt to move an outside pathSeg in the same spine or do the opposite, even when the *Spine* option is enabled in the Selection Options form.

Arguments

g_enable	A Boolean specifying whether the *Stretch* command can move the pathSeg of a malformed spine.

Value Returned

`t`	The *Stretch* command is set to move the pathSeg of a malformed spine case.
`nil`	The operation fails.

Examples

Enables the *Stretch* command to move the pathSeg of a malformed spine case.

```
cleStretchMalformedSpine(t)
```

Disables the *Stretch* command from moving the pathSeg of a malformed spine case.

```
cleStretchMalformedSpine(nil)
```

Related Topics

Selection Options form

Concurrent Layout User Trigger Functions

Use the functions described in this section to register and unregister user-defined triggers and perform other related operations.

- cleGetUserTriggers
- cleRegUserTriggers
- cleUnregUserTriggers

cleGetUserTriggers

```
cleGetUserTriggers(
     [ s_userTriggerSymbol ]
     )
     => r_infoObj / t_procName / nil
```

Description

Retrieves the name of the user-defined callback registered using `cleRegUserTriggers`.

Arguments

s_userTriggerSymbol

User-defined callback whose name you want to retrieve.

Value Returned

r_infoObj	The object containing the user-defined callbacks. Use the following command to query this information: `r_infoObj~>??`
t_procName	Name of the procedure if the user-defined callback is specified.
`nil`	No user-defined callback is registered.

Examples

Returns `nil` because no callback is registered for `cvInitProc`.

```
cleGetUserTriggers(stringToSymbol("cvInitProc"))
=> nil
```

Returns name of the procedure registered for the callback `postImportProc`

```
cleRegUserTriggers(?postImportProc "myTestPostImportProc")
cleGetUserTriggers(stringToSymbol("postImportProc"))
=> "myTestPostImportProc"
```

Returns all registered user-defined callbacks.

```
cleRegUserTriggers(?postInstallProc "myTestPostInstallProc"
?postImportProc "myTestPostImportProc"
?postSipProc "myTestPostSipProc")
```

```
cleGetUserTriggers()~>??
=>(postInstallProc "myTestPostInstallProc" postImportProc "myTestPostImportProc"
postOpenProc nil cvInitProc nil checkCvUpdateProc nil rmbMenuProc nil
hierEditSetupProc nil defLayerProc nil postClearAllProc nil postSipProc
"myTestPostSipProc"
postCreateProc nil postSubmitProc nil postRejectProc nil postCommitProc nil
preOpenScratchProc nil
)
```

Related Topics

cleRegUserTriggers

cleRegUserTriggers

```
cleRegUserTriggers (
     [ ?cvInitProc t_cvInitProc ]
     [ ?postInstallProc t_postInstallProc ]
     [ ?postImportProc t_postImportProc ]
     [ ?postOpenProc t_postOpenProc ]
     [ ?postSipProc t_postSipProc ]
     [ ?checkCvUpdateProc t_checkCvUpdateProc ]
     [ ?hierEditSetupProc t_hierEditSetupProc ]
     [ ?defLayerProc t_defLayerProc ]
     [ ?postClearAllProc t_postClearAllProc ]
     [ ?rmbMenuProc t_rmbMenuProc ]
     [ ?postCreateProc t_postCreateProc ]
     [ ?postSubmitProc t_postSubmitProc ]
     [ ?postRejectProc t_postRejectProc ]
     [ ?postCommitProc t_postCommitProc ]
     [ ?preOpenScratchProc t_preOpenScratchProc ]
     )
     => t / nil
```

Description

Registers user-defined callbacks to customize the cellview for specific Concurrent Layout commands.

Arguments

?cvInitProc *t_cvInitProc*

The user callback to postprocess the cellview after it has been initialized for concurrent layout.

```
cvInitProc(
     r_cxtObj
     )
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout.

You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
```

where the valid value for `cxtObj~>mode` is `cle_manager`.

For example,

```
procedure( myTestCvInitProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode     = %L\n" argCxt~>mode)
 )
cleRegUserTriggers(?cvInitProc "myTestCvInitProc")
```

?postInstallProc *t_postInstallProc*

The user callback to postprocess the concurrent layout cellview after an application is installed using layout editor.

```
postInstallProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
```

where the `cxtObj~>mode` is `cle_manager` or `cle_designer`

For example,

```
procedure( myTestInstallProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window  = %L\n" argCxt~>window)
 info("mode = %L\n" argCxt~>mode)
)
cleRegUserTriggers(?postInstallProc "myTestInstallProc")
```

?postImportProc *t_postImportProc*

The user callback to postprocess the concurrent layout cellview after importing a concurrent layout design partition.

```
postImportProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>figs
cxtObj~>errCxt
```

where

- cxtObj~>mode is cle_manager, or cle_designer
- cxtObj~>*figs* is a list of all modified figures and including new objects from peer design partitions.
- cxtObj~>*errCxt* is context object that contains different objects with edit conflicts.

For example,

```
procedure( myTestImportProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window  = %L\n" argCxt~>window)
 info("mode = %L\n" argCxt~>mode)
 info("figs = %L\n" argCxt~>figs
 info("errCxt = %L\n" argCxt~>errCxt~>??)
)
cleRegUserTriggers(?postImportProc "myTestImportProc")
```

`?postOpenProc` *`t_postOpenProc`*

The user callback to postprocess the concurrent layout cellview after opening the concurrent layout design.

```
postOpenProc(r_cxtObj)
     => t / nil
```

where

`r_cxtObj` is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

`cxtObj~>`*`cellView`*

`cxtObj~>`*`window`*

`cxtObj~>`*`mode`*

where the `cxtObj~>`*`mode`* is `cle_manager` or `cle_designer`

For example,

```
procedure( myTestOpenProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode     = %L\n" argCxt~>mode)
)
cleRegUserTriggers(?postOpenProc "myTestOpenProc")
```

?postSipProc *t_postSipProc*

The user callback to postprocess the concurrent layout cellview after the *Select in Partition* option is enabled.

```
postSipProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>sipMode
```

where

- cxtObj~>*mode* is cle_manager or cle_designer.
- cxtObj~>*sipMode* are disable, select, or enforce.

For example,

```
procedure( myTestSipProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode     = %L\n" argCxt~>mode)
 info("sipMode  = %L\n" argCxt~>sipMode)
)
cleRegUserTriggers(?postSipProc "myTestSipProc")
```

`?checkCvUpdateProc` *`t_checkCvUpdateProc`*

The user callback to postprocess the concurrent layout cellview after opening or refreshing the cellview.

```
checkCvUpdateProc(r_cxtObj)
     => t / nil
```

where

`r_cxtObj` is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>mode
cxtObj~>cellViewOnDisk
```

where

- `cxtObj~>`*`mode`* is `cle_manager` or `cle_designer`
- `cxtObj~>`*`cellViewsOnDisk`* is a list of `libName`, `cellName`, and `viewName` opened or refreshed from the disk.

For example,

```
procedure( myTestCheckCvUpdateProc(argCxt)
 info("mode = %L\n" argCxt~>mode)
 info("cellViewOnDisk = %L\n" argCxt~>cellViewOnDisk)
)
cleRegUserTriggers(?checkCvUpdateProc
"myTestCheckCvUpdateProc")
```

?hierEditSetupProc *t_hierEditSetupProc*

The user callback to preprocess the concurrent layout cellview before you Edit In Place in a concurrent layout design. Return value `t` indicates that Edit In Place is done in Incremental mode. When the return value is `nil`, Edit In Place is done in regular mode.

```
hierEditSetupProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>eipInst
cxtObj~>eipTopCellView
```

where

- `cxtObj~>mode` is `cle_designer`.
- `cxtObj~>eipInst` is the instance name.
- `cxtObj~>eipTopCellView` is the cellview in which you Edit In Place.

For example,

```
procedure( myTestHierEditSetupProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode     = %L\n" argCxt~>mode)
 info("eipInst  = %L\n" argCxt~>eipInst)
info("eipTopCellView = %L\n" argCxt~>eipTopCellView)
 )

cleRegUserTriggers(?hierEditSetupProc
"myTestHierEditSetupProc")
```

?defLayerProc *t_defLayerProc*

The user callback to preprocess the concurrent layout cellview before applying the layers during *Defining Design Partitions*.

```
defLayerProc(r_cxtObj)
     => l_layerName / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>partition
cxtObj~>layerList
```

where

- cxtObj~>*mode* is cle_manager.
- cxtObj~>*partition* is the design partition name.
- cxtObj~>layerList is the list of layer names applied to the design partition.

For example,

The myTestDefLayerProc function only accepts metal layers before applying layers in the design partition.

```
procedure( myTestDefLayerProc(argCxt)
  let((cv layerList filterLayers)
    info("window   = %L\n" argCxt->window)
    info("cellView = %L\n" argCxt->cellView)
    info("mode     = %L\n" argCxt->mode)
    info("partn    = %L\n" argCxt->partition)
    info("layer    = %L\n" argCxt->layerList)
    cv = argCxt->cellView
    foreach(ly argCxt->layerList
      when(techGetLayerFunction(techGetTechFile(cv) ly)
== "metal"
        filterLayers = cons(ly filterLayers)
      )
    )
    reverse(filterLayers)
  )
)
cleRegUserTriggers(?defLayerProc "myTestDefLayerProc")
```

?postClearAllProc *t_postClearAllProc*

The user callback to postprocess the concurrent layout cellview after the *Clear All Design Partitions* command is applied.

```
postClearAllProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>cellViewsToRetain
cxtObj~>cellViewsToCleanUp
cxtObj~>eventType
```

where

- `cxtObj~>mode` is `cle_manager`.
- `cxtObj~>cellViewsToRetain` is the list of `libName`, `cellName`, and `viewName` to be retained after the *Clear All Design Partitions* command is used.
- `cxtObj~>cellViewsToCleanUp` is the list of `libName`, `cellName`, and `viewName` to be removed or reset after the *Clear All Design Partition* command is used.
- `cxtObj~>eventType` is `postClearAll` if the callback is triggered after clearing all design partitions.

For example,

```
procedure( myTestClearAllProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
info("window   = %L\n" argCxt~>window)
info("mode     = %L\n" argCxt~>mode)
 info("cellViewsToRetain = %L\n"
argCxt~>cellViewsToRetain)
 info("cellViewsToCleanUp = %L\n"
argCxt~>cellViewsToCleanUp)
info("eventType = %L\n" argCxt~>eventType)
)
cleRegUserTriggers(?postClearAllProc
"myTestClearAllProc")
```

?rmbMenuProc *t_rmbMenuProc*

The user callback to display the action when you right-click the design partitions in the Concurrent Layout assistant.

```
rmbMenuProc(w_windowId)
     => t / nil
```

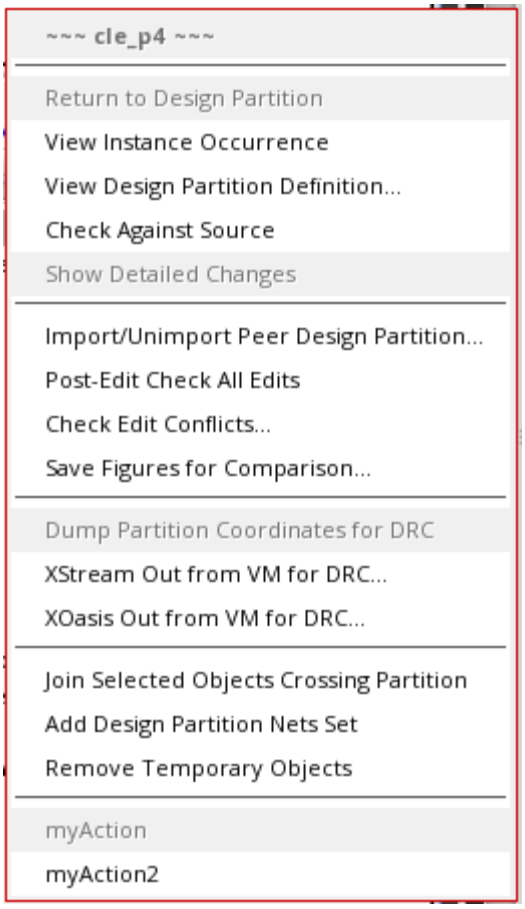
where

w_windowId is the current main canvas area of a session window.

For example,

```
procedure( myUserActions(win)
 list((list("myAction" "Action1CB" t) list("myAction2"
"Action2CB" nil))
)
cleRegUserTriggers(?rmbMenuProc "myUserActions")
```

The example adds `myAction` and `myAction2` commands in the context menu.



?postCreateProc *t_postCreateProc*

The user callback to postprocess the concurrent layout cellview after creating the design partition views.

```
postCreateProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>cellViewsOnDisk
cxtObj~>eventType
```

where

- cxtObj~>*mode* is cle_manager.
- cxtObj~>*cellViewsOnDisk* is the list of libName, cellName, and viewName created on the disk.
- cxtObj~>*eventType* is postCreate if the callback is triggered after creating the design partition views.

For example,

```
procedure( myTestCreateProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode = %L\n" argCxt~>mode)
 info("cellViewsOnDisk = %L\n" argCxt~>cellViewsOnDisk)
 info("eventType = %L\n" argCxt~>eventType)
)
cleRegUserTriggers(?postCreateProc "myTestCreateProc")
```

?postSubmitProc *t_postSubmitProc*

The user callback to postprocess the concurrent layout cellview after submitting the design partitions.

```
postSubmitProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>cellViewsOnDisk
cxtObj~>eventType
```

where

- cxtObj~>*mode* is cle_designer.
- cxtObj~>*cellViewsOnDisk* is the list of libName, cellName, and viewName submitted on the disk.
- cxtObj~>*eventType* value is postSubmit if the callback is triggered after submitting a design partition.

For example,

```
procedure( myTestSubmitProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode = %L\n" argCxt~>mode)
 info("cellViewsOnDisk = %L\n" argCxt~>cellViewsOnDisk)
 info("eventType = %L\n" argCxt~>eventType)
)
cleRegUserTriggers(?postSubmitProc "myTestSubmitProc")
```

?postRejectProc *t_postRejectProc*

The user callback to postprocess the concurrent layout cellview after rejecting a design partition.

```
postRejectProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>cellViewsOnDisk
cxtObj~>eventType
cxtObj~>comment
```

where,

- cxtObj~>*mode* is cle_manager.
- cxtObj~>*cellViewsOnDisk* will be the list of libName, cellName, and viewName rejected on disk.
- cxtObj~>*evenType* is postReject if the callback is triggered after rejecting the design partitions.
- cxtObj~>comment is the reason user specifies when rejecting a design partition.

For example,

```
procedure( myTestRejectProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode = %L\n" argCxt~>mode)
 info("cellViewsOnDisk = %L\n" argCxt~>cellViewsOnDisk)
 info("eventType = %L\n" argCxt~>eventType)
 info("rejected reason = %L\n" argCxt~>comment)
 )
 cleRegUserTriggers(?postRejectProc "myTestRejectProc")
```

?postCommitProc *t_postCommitProc*

The user callback to postprocess the concurrent layout cellview after committing a design partition.

```
postCommitProc(r_cxtObj)
     => t / nil
```

where

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>cellViewsOnDisk
cxtObj~>eventType
```

where

- cxtObj~>*mode* is cle_manager.
- cxtObj~>*cellViewsOnDisk* is a list of libName, cellName, and viewName committed on the disk.
- cxtObj~>*eventType* is postCommit if the callback is triggered after committing design partitions.

For example,

```
procedure( myTestCommitProc(argCxt)
 info("cellView = %L\n" argCxt~>cellView)
 info("window   = %L\n" argCxt~>window)
 info("mode = %L\n" argCxt~>mode)
 info("cellViewsOnDisk = %L\n" argCxt~>cellViewsOnDisk)
 info("eventType = %L\n" argCxt~>eventType)
 )


cleRegUserTriggers(?postCommitProc "myTestCommitProc")
```

?preOpenScratchProc *t_preOpenScratchProc*

The user callback to preprocess the top cellview before opening a concurrent layout scratch cellview for editing. In this case, the top cellview is not locked but write permission is required for corresponding layout.oa.

In most DM environments, you still have the write permission after checkin; otherwise, you can use this callback for non-exclusive checkout of the top cellview.

```
preOpenScratchProc(r_cxtObj)
```

```
=> t / nil
```

where,

r_cxtObj is the object that retrieves the additional information related to the design open in concurrent layout. You can retrieve the information in the following way:

```
cxtObj~>cellView
cxtObj~>window
cxtObj~>mode
cxtObj~>cellViewsOnDisk
cxtObj~>topCellViewOnDisk
```

where,

- `cxtObj~>mode` is `cle_designer`. The manager has no need to edit a scratch file.
- `cxtObj~>cellViewsOnDisk` is a list of `libName`, `cellName`, and `viewName` opened on the disk.
- `cxtObj~>topCellViewOnDisk` is the top cellview of `libName`, `cellName`, and `viewName` on which the concurrent layout scratch cellview is based.

For example,

```
procedure( myTestPreOpenScratchProc(argCxt)
info("cellView = %L\n" argCxt->cellView)
info("window   = %L\n" argCxt->window)
info("mode     = %L\n" argCxt->mode)
info("cellViewsOnDisk = %L\n" argCxt->cellViewsOnDisk)
info("top cellView = %L\n" argCxt->topCellViewOnDisk)
)
cleRegUserTriggers(?preOpenScratchProc
"myTestPreOpenScratchProc")
```

Value Returned

`t`	The specified user-defined callback function was registered successfully.
`nil`	The user-defined callback function was not registered.

cleUnregUserTriggers

```
cleUnregUserTriggers(
     [ l_userTriggerSymbols ]
     )
     => t/nil
```

Description

Unregisters all current or specified user-defined callback functions.

Arguments

l_userTriggerSymbols

List of registered user-defined callback functions.

Value Returned

`t`	The specified user-defined callback functions were unregistered.
`nil`	The operation failed because no user-defined callback function was registered.

Example

All user-defined callback functions are unregistered.

```
cleUnregUserTriggers()
=> t
```

The specified user-defined callback functions `'cvInitProc`, `'postOpenProc`, and `'postImportProc` are unregistered.

```
cleUnregUserTriggers(list('cvInitProc 'postOpenProc 'postImportProc))
=> t
```

22

Process Design Kit (PDK) Cockpit Functions

This topic provides a list of Cadence® SKILL functions associated with Process Design Kit (PDK) Cockpit.

- pdkeqCockpit

pdkeqCockpit

```
pdkeqCockpit(
     g_activate
     )
     => t / nil
```

Description

Launches the PDK cockpit.

Arguments

None

Arguments

None

Example

```
pdkeqCockpit()
```

Launches the PDK cockpit.