cadence®

Virtuoso Schematic Editor 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

1

Virtuoso Schematic Editor Human Interface (HI) Functions

The Virtuoso® Schematic Editor SKILL human interface (HI) functions and arguments let you customize the schematic editor menus and bindkeys. They are designed to be top-level functions that are not called by other functions. Human interface functions are not intended for procedural use.

Most Virtuoso schematic editor human interface functions begin with `schHi`. You can often derive the name of the HI function from the PI function by adding the `Hi` portion of the name. For example, the PI function

```
schCreateWire
```

has the corresponding HI function

```
schHiCreateWire
```

Most HI functions are interactive, requiring you to interact with a form or prompting you to click an object or a location in your schematic. The functions accept input from an options form associated with the corresponding menu command. When you call a function without specifying any required arguments, an option form automatically appears. For example, when you type `schHiCopy` in the CIW, the system prompts you to use your cursor to point to the object in your schematic that you want to copy. Option forms provide a graphical interface that lets you specify function arguments. You can manually open an option form for an active command by pressing the `F3` function key on your keyboard.

Most interactive functions remain active until you explicitly cancel them or until you start a new function. Some interactive functions, such as copy, return the action immediately even though the command is still active. These functions are based on enter functions. All HI functions return a Boolean value, either `t` or `nil`. When the function completes normally, the function returns a `t`; when the function fails or is canceled, the function returns a `nil`. In this way, you can create compound functions that take error recovery action.

Some functions are restricted to either schematic or schematic symbol view types. Other functions are restricted to multisheet designs, indexes, or sheets.

Related Topics

Virtuoso Schematic Editor Procedural Interface (PI) Functions

annToggleInfoBalloonVisibilityStatus

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

Description

Toggles the visibility of the annotation balloons on the schematic canvas. Also, displays and removes the pinned balloons on all open schematic windows.

Arguments

None

Value Returned

`t`	The annotation balloons are displayed or removed from the schematic canvas.
`nil`	The command is unsuccessful or an error occurred.

Examples

```
annToggleInfoBalloonVisibilityStatus()
=> t
```

annPinCurrentInfoBalloon

```
annPinCurrentInfoBalloon(
     t_windowId
     )
     => t / nil
```

Description

Pins the current unpinned annotation balloon on the specified schematic window.

Arguments

t_windowId	The ID of the window on which you want to pin the balloons.

Value Returned

t	The annotation balloons on the specified window are pinned.
nil	The command is unsuccessful or an error occurred.

Examples

```
annPinCurrentInfoBalloon(window(2))
=> t
```

cdsName

```
cdsName(
      )
```

Description

The `cdsName` function uses information about the instance currently being drawn in order to compute the data returned by the schematic editor during a redraw. Consequently, `cdsName` is only intended to be called as the value of an `ILLabel` on a symbol, and is not intended for direct evaluation during a program. If it is invoked directly, information on the current drawn instance will not be available. The `cdsName` will display label information, usually placed on the layer `annotate drawing7`, near the cell name or instance name.

Note: This can also be done using the *Virtuoso Symbol Editor's Create – Label* option, and choosing *Annotate Instance Label* in the *Label Choice* field.

This function is also attached to a cell symbol view when you use the Add Symbol Label form and the Annotation Setup Form is used to control the display of this label type.

The CDF *Interpreted Labels Information* sections of both the library and component are also used to configure what the label displays. The pertinent *Interpreted Labels Information* parameters are `instDisplayMode` and `instNameType`.

Arguments

None

Value Returned

None

cdsNetExpr

```
cdsNetExpr(
     )
```

Description

The `cdsNetExpr` function is used inside the `ILLabel` that is created using the *Create – Net Expression* command. This function uses information about the instance currently being drawn in order to compute the data returned by the schematic editor during a redraw. Consequently, `cdsNetExpr` is only intended to be called as the value of an `ILLabel` on a symbol, and is not intended for direct evaluation during a program. If it is invoked directly, information on the current drawn instance will not be available.

Arguments

None

Value Returned

None

cdsParam

```
cdsParam(
     n_index
     )
```

Description

The `cdsParam` function displays label information, usually placed on the layer `annotate drawing`, about the parameter values or backannotated parameter values. This function uses information about the instance currently being drawn in order to compute the data returned by the schematic editor during a redraw. Consequently, `cdsParam` is only intended to be called as the value of an `ILLabel` on a symbol, and is not intended for direct evaluation during a program. If it is invoked directly, information on the current drawn instance will not be available.

The `cdsParam` function is also attached to a cell symbol view when you use the Add Symbol Label form. The Annotation Setup Form is used to control the display of this label type.

The CDF *Interpreted Labels Information* sections of both the library and component are also used to configure what this label displays. You can select the parameters to be displayed, the order in which they are listed, and whether their values are displayed. For more details, see *Specifying cdsParam Parameters to Display* in the Component Description Format User Guide.

The pertinent *Interpreted Labels Information* parameters are `paramDisplayMode`, `paramLabelSet`, `opPointLabelSet`, `modelLabelSet`, `paramEvaluate`, and `paramSimType`.

Arguments

n_index

Three labels are usually generated during automatic symbol generation, but you can define additional labels. The only requirement for the parameter labels is that you number them sequentially, starting with 1.

Note: By default the parameter name and its value are displayed in the following format when a value has been explicitly set on the instance,

```
parameter name = value (w=7u)
```

However, if there is no value stored on the instance but it is picking up the CDF default, the format changes so that = is replaced with : as follows,

```
parameter name : value (w:7u)
```

Value Returned

None

cdsTerm

```
cdsTerm(
     s_pinName
     )
```

Description

The function `cdsTerm` displays label information, usually placed on the layer `annotate drawing8`, near the pin or a net attached to the pin. It is also attached to a cell symbol view when you use the Add Symbol Label form, and the Annotation Setup Form is used to control the display of this label type.

The `cdsTerm` function uses information about the instance currently being drawn in order to compute the data returned by the schematic editor during a redraw. Consequently, `cdsTerm` is only intended to be called as the value of an `ILLabel` on a symbol, and is not intended for direct evaluation during a program. If it is invoked directly, information on the current drawn instance will not be available.

The CDF *Interpreted Labels Information* sections of both the library and component are also used to configure what this label displays. The pertinent *Interpreted Labels Information* parameters are `termDisplayMode`, `termSimType`, and `netNameType`.

Arguments

s_pinName	If the symbol contains special characters, you must put the string in quotation marks or escape the special characters properly.

Value Returned

None

heHiEditConfig

```
heHiEditConfig(
     )
     => t
```

Description

Opens the hierarchy editor if the current editing window has a design opened within the context of a configuration.

Arguments

None

Value Returned

Always returns `t`.

heHiSetInstBinding

```
heHiSetInstBinding(
     )
     => t
```

Description

Sets the instance bindings of an instance in a cellview of open configurations. The form associated with this function is updated with current binding information. The new bindings are communicated to the hierarchy editor.

Arguments

None

Value Returned

Always returns `t`.

heHiShowViewsFound

```
heHiShowViewsFound(
     )
     => t
```

Description

Shows the current view being used for each instance in a form.

Arguments

None

Value Returned

Always returns `t`.

heHiUpdate

```
heHiUpdate(
     )
     => t
```

Description

Updates the information in the hierarchy editor after you edit the configuration.

Arguments

None

Value Returned

Always returns `t`.

hiPrevWinView

```
hiPrevWinView(
     )
```

Description

Scrolls back through up to ten window zoom or pan views.

Arguments

None

Value Returned

None

hiNextWinView

```
hiNextWinView(
     )
```

Description

Scrolls forward through up to ten window zoom or pan views.

Arguments

None

Value Returned

None

schAddSelectPt

```
schAddSelectPt(
     )
     => t
```

Description

Selects the object under the cursor. Maintains the selected set and adds the object to the selected set. Usable when editing schematics. Is equivalent to the Graphics Editor `mouseAddSelectPt` function. Provides other functions with the identity of the most recently selected object, which is required for extended selection.

Arguments

None

Value Returned

Always returns `t`.

schDirectEdit

```
schDirectEdit(
     x_index
     )
     => t
```

Description

Directly edits or manipulates the object under the cursor. Usable when editing schematics and symbols.

If the object is under the cursor or if the object is in the selected set, you can modify all objects in the selected set. If the object under the cursor is not in the selected set, you can modify only that object. If there is no object under the cursor, the `SelectByArea` process is used.

In the `schBindkey.il` file, `DrawThru1` is bound to `schDirectEdit` as shown:

```
<DrawThru1> schDirectEdit(1)
Shift<DrawThru1> schDirectEdit(2)
Ctrl<DrawThru1> schDirectEdit(3)
```

The following table shows what kind of edit takes place.

Object Type/Index	1	2	3
Instance (or Block)	Stretch	Copy	Move
Wire	Stretch	Copy	Move
Wire Vertex	Stretch	Add Wire	Move
Schematic Pin	Stretch	Add Wire	Move
Note Shape	Move	Copy	Move
Note Shape Edge	Stretch	Move	Move
Note Shape Vertex	Stretch	Move	Move
Symbol Pin	Move	Add Line	Move
Labels	Move	Copy	Move
Instance Pin	Add Wire	Add Wire	Add Wire
Instance Label	Move	Move	Move
Net Expression	Move	Copy	Move

Arguments

x_index	An integer that specifies which function to execute. Valid Values: `1`, `2`, `3`

Value Returned

Always returns `t`.

Examples

If the cursor is not over an object, executes `selectByArea`. If the object under the cursor is an instance, executes `stretch` on the instance. If that instance belongs to the selected set, stretches all objects in the selected set.

```
schDirectEdit( 1 )
```

If the cursor is not over an object, executes an additive `selectByArea`. If the object under the cursor is an instance, executes `copy` on the instance. If that instance belongs to the selected set, copies all objects in the selected set.

```
schDirectEdit( 2 )
```

If the cursor is not over an object, executes `deselectByArea`. If the object under the cursor is an instance, executes `move` on the instance. If the instance belongs to the selected set, moves all objects in the selected set.

```
schDirectEdit( 3 )
```

Related Topics

Bindkeys and Access Keys

schExtendSelectPt

```
schExtendSelectPt(
     )
     => t
```

Description

Extends the selection of the object under the current cursor position by selecting objects around the current object. Usable only when editing schematics.

Searches through the schematic cellview for objects that are physically touching the object under the cursor and adds them to the selected set. If the cursor is over an object, this function selects the object. If the object is already selected, this function extends the selection. This function adds any objects in the next selection level to the selected set. It increments the selection level until something is selected. When this function reaches the maximum selection level, it cycles back to the single object. For example,

- Extending a wire selects all segments in the same branch; selection stops at T-intersections, pins, instance pins, and changes in wire width. Executing the function a second time selects all connected wire segments, stopping only at pins and instance pins.
- Extending an instance selects all single wire segments connected to any of its instance pins. Repeating the function extends along wires as described above.
- Extending a label selects its owner. Repeating the function extends the owner as described above for wires, pins, and instances.

Arguments

None

Value Returned

Always returns `t`.

schHiAbout

```
schHiAbout(
     )
     => t
```

Description

Opens the product information window, which indicates the schematic editor release number and copyright information.

Arguments

None

Value Returned

Always returns `t`.

schHiAlternateView

```
schHiAlternateView(
     )
     => t
```

Description

Changes the view name of a single component by toggling through the list of possible views. Usable only while editing a schematic cellview. Operates on one instance at a time.

If the selected set contains only one instance, that instance is modified. If no instances are in the selected set, you are prompted to point at an object to modify. If more than one instance is in the selected set, you are prompted to select only one instance. The set of view names is derived from the views that exist for the instance's master excluding those views referenced in the `schCycleViewNameExclusionList` global variable defined in the `schConfig.il` file.

Arguments

None

Value Returned

Always returns `t`.

schHiAlign

```
schHiAlign(
     s_justify
     [ g_windowId ]
     )
     => t / nil
```

Description

Aligns the objects on the canvas. If the objects are already selected on the canvas, the pre-select mode is used; otherwise, the post-select mode is used for aligning the objects.

Arguments

s_justify	The direction in which the objects need to be aligned. Possible values are `left`, `right`, `top`, `bottom`, `vertical`, and `horizontal`.
g_windowId	The window indicating the cellview where you want to start aligning. If not specified, the current window is used. This is an optional argument.

Value Returned

`t`	Objects were aligned successfully.
`nil`	Objects could not be aligned.

Examples

```
schHiAlign( 'left )
schHiAlign( 'right hiGetCurrentWindow() )
schHiAlign( 'top window(3) )
```

Related Topics

Aligning

schHiCellViewProperty

```
schHiCellViewProperty(
     )
     => t
```

Description

Displays the options form showing the properties for the current cellview. Usable only when editing schematic or symbol cellviews.

Arguments

None

Value Returned

Always returns `t`.

schHiChangeEditMode

```
schHiChangeEditMode(
     t_newMode
     )
     => t
```

Description

Sets the mode for the design in the current window to read or append. The mode is the same as that supplied to `dbOpenCellViewByType`.

To prevent any unintentional loss of data, you should not use this function to change the mode to overwrite mode.

Arguments

t_newMode	New access mode; must be enclosed in quotation marks. Valid Values: `r` (read), `a` (append)

Value Returned

Always returns `t`.

Examples

Changes the mode to read and returns a `t`.

```
schHiChangeEditMode( "r" )
```

schHiCheck

```
schHiCheck(
     [ ?action t_action ]
     )
     => t
```

Description

Checks the connectivity of a schematic and optionally starts the schematic rules checker (SRC) or the cross-view checker (VIC). Usable only when editing schematics. A dialog box shows the total number of errors and warnings detected when the function is complete. The schematic can be read-only or editable.

Arguments

`?action` *`t_action`*	Defines the action to take; must be enclosed in quotation marks. Valid Values: `run`, `editOptions`, `editOptionsAndRun` Default: `editOptionsAndRun`

Value Returned

Always returns `t`.

Examples

Displays the options form for modifying the check options.

```
schHiCheck( ?action "editOptions" )
```

Runs the checks that are set on the form.

```
schHiCheck( ?action "run" )
```

Lets you modify the various check option settings on a form. The check is then performed on the schematic in the current window.

```
schHiCheck( ?action "editOptionsAndRun" )
```

Related Topics

schHiCheckHier

schHiCheckAndSave

```
schHiCheckAndSave(
     )
     => t
```

Description

Performs the checks specified by the check options and saves the schematic to disk under the same cell name and view name, and in the same library. Usable only when editing schematics.

Provides a simple interface to the schematic check function and saves the schematic if no connectivity errors are encountered during the check. If errors do exist, then depending on the *Check and Save Action on Error* setting, the schematic is either saved or not saved or you are prompted for the next action to perform.

Arguments

None

Value Returned

Always returns `t`.

schHiCheckHier

```
schHiCheckHier(
     [ ?action t_action ]
     [ ?refLibs t_refLibs ]
     )
     => t
```

Description

Performs the specified checks on the current schematic and the hierarchy below it. Also updates the connectivity as needed and runs the schematic rules checker (SRC), the cross-view checker (VIC), or both. Usable only when editing schematics. Only processes schematics found in the hierarchy starting from the current cellview. The view name list used to control the traversal is taken from the window in which the function is run.

Arguments

`?action` *`t_action`*	Defines the action to take; must be enclosed in quotation marks. Valid Values: `run`, `editOptions`, `editOptionsAndRun` Default: `editOptionsAndRun`
`?refLibs` *`t_refLibs`*	Additional reference libraries to process; must be enclosed in quotation marks.

Value Returned

Always returns `t`.

Examples

Displays the form for modifying the check hierarchy options.

```
schHiCheckHier( ?action "editOptions" )
```

Runs the hierarchical check with the options set as they are on the form; the empty string for the *`t_refLibs`* argument specifies that no reference libraries are to be checked.

```
schHiCheckHier( ?action "run" ?refLibs " ")
```

Lets you modify the various check option settings on a form.

```
schHiCheckHier( ?action "editOptionsAndRun" )
```

schHiCloneSymbol

```
schHiCloneSymbol(
     [ ?libraryName t_libraryName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     )
     => t
```

Description

Copies graphics from an existing symbol library into the symbol design you are currently editing. Usable only when editing symbols. If you do not specify any argument, the options form appears and prompts you for the values of these fields.

Arguments

`?libraryName` *`t_libraryName`*

Library that contains the symbol you want to clone; must be enclosed in quotation marks.

`?cellName` *`t_cellName`*

Name of the cell you want to clone; must be enclosed in quotation marks.

`?viewName` *`t_viewName`*

Name of the view you want to clone; must be enclosed in quotation marks.

Value Returned

Always returns `t`.

Examples

Clones the graphic of the `nand2` symbol from the `sample` library and prompts you for a point to place it in your current symbol.

```
schHiCloneSymbol( ?libraryName "sample" ?cellName "nand2" ?viewName "symbol" )
```

schHiComputePinRef

```
schHiComputePinRef(
     [ ?reportFile t_reportFile ]
     [ ?display t_display ]
     [ ?formatString t_formatString ]
     [ ?reportDups t_reportDups ]
     [ ?sortByDir t_sortByDir ]
     )
     => t
```

Description

Displays the Cross-Reference Options form for current index or sheet schematic, which computes, stores, and lists zone references for all pins and offsheet connectors in a multisheet schematic. The zone references identify where pins on other sheets reference the same net. Uses stored references to identify pin locations. The pin references can be displayed in the schematic alongside each pin or written to a report file. Can be used only when editing either the index or a sheet of a multisheet design. The index requires checking before zones can be computed.

Arguments

`?reportFile` *`t_reportFile`*

Filename for the cross-reference report; must be enclosed in quotation marks. Use an empty string to suppress the report file.
Default: `" "`

`?display` *`t_display`*

Controls display of cross-references in the schematic; must be enclosed in quotation marks. Set this argument to `on` to display cross-references or to `off` to remove any existing cross-references.
Default: `on`

`?formatString` *`t_formatString`*

Controls the cross-reference format. You can build the cross-reference format using any combination of the following in any order:

`sheetNumber zone referenceName direction`

Default: `schGetEnv("pinRefFormat")`

`?reportDups` *`t_reportDups`*

Controls reporting of duplicate pin references found within the same zone; must be enclosed in quotation marks. Set this argument to `on` to report duplicate pin references or to `off` to suppress these reports.
Default: `schGetEnv("pinRefDuplicates")`

`?sortByDir` *`t_sortByDir`*

Controls sorting of pin references; must be enclosed in quotation marks. Set this argument to `on` to sort by direction or to `off` to sort by sheet number.
Default: `schGetEnv("pinRefSorting")`

Value Returned

Always returns `t`.

Examples

Displays the Cross-Reference Options form and also creates a report file named `design.xref`.

```
schHiComputePinRef( ?reportFile "design.xref" )
```

Displays the Cross-Reference Options form to produce cross-references in the schematic alongside each pin in sheet `1` zone `B`. Duplicate references are not reported and references are sorted by direction.

```
schHiComputePinRef( t_reportFile "" ?display "on" ?formatString "1 B" ?display
"off" ?display "on" )
```

schHiCopy

```
schHiCopy(
     [ ?formFlag g_formFlag ]
     [ ?rows x_numrows ]
     [ ?columns x_numcols ]
     [ ?useSelSet t_useSelSet ]
     )
     => t
```

Description

Copies objects and data such as object properties. Objects can be copied between different schematic cellviews. Usable when editing schematics or symbols.

If the selected set contains multiple objects, you are prompted to click a reference point.

Note: If the schematic cellview object selection set (`schSelSet`) contains partially selected objects, these objects will be excluded from the selection set to be copied.

Arguments

`?formFlag` *`g_formFlag`*

Specifies whether or not to bring up the options form. A `t` displays the options form. A `nil` copies any selected object using default values set on the form.

`?rows` *`x_numrows`*

Number of rows to generate. Range is limited by the system. If *`x_numrows`* is greater than `1`, you are prompted to select a destination for the first copy in the second row. The schematic editor calculates the offsets for placing the remaining elements of the array. The resultant objects are copies of the original and are not stored as an array. *`x_numrows`* resets to `1` when you complete the copy.
Default: `1`

`?columns` *`x_numcols`*

Number of columns to generate. Range is limited by the system. If *`x_numcols`* is greater than `1`, you are prompted to click at a location for the second copy in the first row. The schematic editor calculates the offsets for placing the remaining elements of the array. The resultant objects are copies of the original and are not stored as an array. *`x_numcols`* resets to `1` when you complete the copy.
Default: `1`

`?useSelSet` *`t_useSelSet`*

Specifies whether to copy the entire selected set or only the object you select with the mouse; must be enclosed in quotation marks. If you set this argument to `noSelSet`, any previously selected objects are ignored and the function is nonmodal. If you set the argument to `useSelSet`, any previously selected objects are used and the function is nonmodal; otherwise, the function is modal and prompts you for objects to copy.
Valid Values: `useSelSet`, `noSelSet`
Default: `useSelSet`

Value Returned

Always returns `t`.

Examples

Makes six copies of objects you select and calculates the placement of each object according to your selected destination point of the first copy in the second row and the second copy in the first row. The `nil` value does not bring up the options form but uses the current field values.

```
schHiCopy( ?formFlag nil ?rows 2 ?columns 3 )
```

Displays the options form you use to specify the values for your copy.

```
schHiCopy( ?formFlag t )
```

Ignores the selected set and prompts you to select the object to copy.

```
schHiCopy( ?formFlag nil ?rows 1 ?columns 1 ?useSelSet "noSelSet" )
```

schHiCreateBlockInst

```
schHiCreateBlockInst(
     [ ?libraryName t_libraryName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     [ ?blockSampleName t_blockSampleName ]
     [ ?instanceName t_instanceName ]
     [ ?pinNameSeed t_pinNameSeed ]
     )
     => t
```

Description

Creates a block and places an instance of a block in a schematic. Usable only when editing schematics. If you do not type a library, cell, view, and instance name as arguments, the options form appears and prompts you for these values. The block choices are defined by the `schBlockTemplate` variable in the `schConfig.il` file. If you type freeform, the schematic editor prompts you to type a rectangular shape by clicking on two points of the rectangle.

Arguments

`?libraryName` *`t_libraryName`*

An existing library in which you want to create your block; must be enclosed in quotation marks.

`?cellName` *`t_cellName`*

Cell name of the block you want to create; must be enclosed in quotation marks.

`?viewName` *`t_viewName`*

View you want to create; must be enclosed in quotation marks.

`?blockSampleName` *`t_blockSampleName`*

Name of a block sample that represents the boundary of the block; must be enclosed in quotation marks.
Valid Values: `small`, `medium`, `large`, `2 by 1`, `1 by 2`, `alu`, `mux4`, `mux8`

`?instanceName` *`t_instanceName`*

Unique name to assign to your instance; must be enclosed in quotation marks.

`?pinNameSeed` *`t_pinNameSeed`*

Seed name that the schematic editor uses when generating names of new pins created on this block; must be enclosed in quotation marks.

Value Returned

Always returns `t`.

Examples

Creates a new block with the specified fields. Prompts you to type a rectangular shape that specifies the shape of the symbol as well as the origin of the instance of the new block. *`t_pinNameSeed`* is set to `pin` and *`t_instanceName`* is assigned a unique value.

```
schHiCreateBlockInst( ?libraryName "ASIC_LIB" ?cellName "BLOCK1" ?viewName
"SYMBOL_NEG" )
```

Creates two blocks named `BLOCK1` and `BLOCK2`. Both blocks are created in the `ASIC_LIB` library and the view name is `SYMBOL_NEG`. The instance names are `I1` and `I2`. The boundary of the blocks is a fixed size based on the pointlist in the `blockSample` map that relates the `large blockSampleName`. The schematic editor drags the first block and prompts you for a location to place the instance. It then drags the second block and prompts you for a location.

```
schHiCreateBlockInst( ?libraryName "ASIC_LIB" ?cellName "BLOCK1 BLOCK2" ?viewName
"SYMBOL_NEG" ?blockSampleName "large" ?instanceName "I1 I2" ?pinNameSeed "PIN" )
```

schHiCreateInst

```
schHiCreateInst(
     [ ?libraryName t_libraryName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     [ ?instanceName t_instanceName ]
     [ ?rows x_rows ]
     [ ?columns x_columns ]
     )
     => t
```

Description

Places an instance of a cellview in a schematic. Usable only when editing schematics.

Arguments

`?libraryName` *`t_libraryName`*

Library that contains the cellview; must be enclosed in quotation marks.

`?cellName` *`t_cellName`*

Cell name you want to use; must be enclosed in quotation marks.

`?viewName` *`t_viewName`*

View you want to use; must be enclosed in quotation marks.

`?instanceName` *`t_instanceName`*

Unique name to assign to your instance; must be enclosed in quotation marks. To name more than one instance, use a space as a delimiter. The schematic editor places the instances in the order in which you specified them.

?rows *x_rows*	Number of rows of instances to create. Range is limited by the system. If *x_rows* is greater than 1, you are prompted to click at a location to place the first instance in the second row. The schematic editor places the first components of the remaining columns, then drags all first row components and places all remaining components. It names each instance with the names that you specified in *t_instanceName*. If *t_instanceName* is empty, unique names are generated for each instance placed.
?columns *x_columns*	Number of columns of instances to create. Range is limited by the system. If *x_columns* is greater than 1, you are prompted to click at a location to place the first instance in the second column. The schematic editor places the first components of the remaining columns, then drags all first row components and places all remaining components. It names each instance with the names that you specified in *t_instanceName*. If *t_instanceName* is empty, unique names are generated for each instance placed.

Value Returned

Always returns t.

Examples

Drags the requested AND2 symbol from the SAMPLE library until you click to place it. Symbol view name is SYMBOL and the instance name is IO.

```
schHiCreateInst( ?libraryName "SAMPLE" ?cellName "AND2" ?viewName "SYMBOL"
?instanceName "IO" )
```

Creates an array of 12 components. Names the first two instances in the first row A and B. Automatically generates unique instance names for the remaining instances in the array.

```
schHiCreateInst( ?libraryName "SAMPLE" ?cellName "AND2" ?viewName "SYMBOL" "A B"
4 3 )
```

schHiCreateInstBox

```
schHiCreateInstBox(
     [ ?autoBox g_autoBox ]
     )
     => t
```

Description

Creates an instance box for the symbol you are editing. An instance box defines a rectangular region in which an instance of a symbol is selectable. Usable only when editing symbols.

If you specify *g_autoBox*, the editor creates an instance box. Otherwise, you are prompted to type a rectangle to represent the instance box.

Arguments

?autoBox *g_autoBox*	Specifies whether the instance box is created automatically (`t`) or manually (`nil`).

Value Returned

Always returns `t`.

Examples

Calculates the size of the rectangle to represent the instance box by determining the centers of all symbol pins, device shapes, and device labels.

```
schHiCreateInstBox( ?autoBox t )
```

schHiCreateKanjiSymbol

```
schHiCreateKanjiSymbol(
     [ ?kanjiFile t_kanjiFile ]
     [ ?libName t_libName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     )
     => t / nil
```

Description

Creates a symbol view containing kanji characters based on the information in an EUC-packed format kanji file. The software generates a symbol cellview that contains kanji character graphics.

Arguments

`?kanjiFile` *`t_kanjiFile`*
: Filename of a preexisting EUC-packed kanji file; must be enclosed in quotation marks.

`?libName` *`t_libName`*
: Library name of the created symbol; must be enclosed in quotation marks.

`?cellName` *`t_cellName`*
: Cell name of the created symbol; must be enclosed in quotation marks.

`?viewName` *`t_viewName`*
: Viewname of the created symbol; must be enclosed in quotation marks.

Value Returned

`t`
: The Kanji symbol was created.

`nil`
: Caller supplied optional arguments that are not a string.

Examples

Creates a symbol from the Japanese EUC kanji input file named `kanji/kanjitxt.euc` and stores the output symbol in `Class adder4bit_kanji_9`.

```
schHiCreateKanjiSymbol("kanji/kanjitxt.euc" "Class" "adder4bit_kanji_9" "symbol")
```

schHiCreateMappingSchematic

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

Description

Displays the Create a Mapping Schematic form.

Arguments

None

Value Returned

`t`	Mapping schematic form displayed.
`nil`	Command failed or returned an error.

Related Topics

Creating a Mapping Schematic

schHiCreateNetExpression

```
schHiCreateNetExpression(
     [ ?netExpr t_netExpr ]
     [ ?justify t_justify ]
     [ ?fontStyle t_fontStyle ]
     [ ?fontHeight n_fontHeight ]
     )
     => t
```

Description

Creates an inherited connection and the corresponding net expression label. Usable when editing schematics or symbols.

If you do not specify *t_netExpr* or if you specify it as `nil`, the options form appears and prompts you for the net expression. If *t_justify*, *t_fontStyle*, and *n_fontHeight* are not specified, the software applies the current value of the respective environment variables: `createLabelJustify`, `createLabelFontStyle`, and `createLabelFontHeight`. The editor drags the label described by arguments and prompts you to select a location to place the label.

Arguments

`?netExpr` *t_netExpr*	A string containing the net expression in NLP syntax; must be enclosed in quotation marks.
`?justify` *t_justify*	Justification to give the net expression label text with respect to its placement; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
`?fontStyle` *t_fontStyle*	Label font style; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`
`?fontHeight` *n_fontHeight*	Label height in user units. Default: `0.0625`

Value Returned

Always returns `t`.

Examples

Creates the net expression label `"gnd!"`.

```
schHiCreateNetExpression( "[@gnd:%:gnd!]" "lowerLeft" "stick" 0.0625 )
```

Related Topics

The Syntax of an Inherited Net Expression

schHiCreateNoteLabel

```
schHiCreateNoteLabel(
     [ ?text t_text ]
     [ ?justify t_justify ]
     [ ?fontStyle t_fontStyle ]
     [ ?fontHeight n_fontHeight ]
     [ ?type t_type ]
     )
     => t
```

Description

Creates a note label to annotate the design for documentation purposes. These shapes do not affect connectivity. Usable when editing schematics or symbols.

If you do not specify *t_text* or you specify it as `nil`, the options form appears and prompts you for the note label text. The editor drags the label described by arguments and prompts you to select a location to place the label.

Arguments

`?text` *t_text*	Text for your note to include spaces, tabs, and new lines; must be enclosed in quotation marks.
`?justify` *t_justify*	Justification of the label text with respect to its placement; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
`?fontStyle` *t_fontStyle*	Label font style; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`
`?fontHeight` *n_fontHeight*	Label height in user units. Default: `0.0625`
`?type` *t_type*	Label type; must be enclosed in quotation marks. Valid Values: `normalLabel`, `NLPLabel`, `ILLabel` Default: `normalLabel`

Value Returned

Always returns `t`.

Examples

Creates the single string label `Low Pass Filter Section` using the current settings of all other arguments.

```
schHiCreateNoteLabel( ?text "Low Pass Filter Section" )
```

schHiCreateNoteShape

```
schHiCreateNoteShape(
     [ ?shape t_shape ]
     [ ?style t_style ]
     [ ?width n_width ]
     )
     => t
```

Description

Creates a shape to annotate the design for documentation purposes. These shapes do not affect connectivity. Usable when editing schematics or symbols.

Options forms are seeded with the argument values. If the `style` argument is omitted for any shape, the options form is displayed prompting you to specify the missing values. This also occurs if you omit the `width` argument when creating a `line` shape.

Arguments

`?shape` *`t_shape`*	Shape to create; must be enclosed in quotation marks. Valid Values: `line`, `rectangle`, `polygon`, `circle`, `ellipse`, `arc`
`?style` *`t_style`*	Line style of the shape; must be enclosed in quotation marks. Valid Values: `solid`, `dashed`
`?width` *`n_width`*	The width of the line, if *`t_shape`* is set to `line`. If the width is zero, this command creates narrow lines using the minimum possible width. If the width is set to anything other than zero, this command creates wide lines with a configurable width. **Note:** You can modify the way that all narrow note lines are displayed by editing the Data Registry File preferences for the (text, drawing) LPP.

Value Returned

Always returns `t`.

Examples

Starts creating a rectangle using a solid line in the current window.

```
schHiCreateNoteShape( ?shape "rectangle" ?style "solid" )
```

Starts creating a solid line with a width of 0.0625 in the current window.

```
schHiCreateNoteShape( ?shape "line" ?style "solid" ?width "0.0625" )
```

schHiCreatePatchcord

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

Description

Displays the Create Patchcord form where you can create, and add, a `patchCord` instance on a schematic along with appropriately named net stubs.

The expected use is to allow for simple aliasing of nets, where you can enter an alias prefix, for example `cd`, and also the design nets that will be used in that alias, for example `ctrl`, `d<0:4>` etc. The `patchCord` will then, using this example, have the nets `cd<0:5>` and `ctrl`, `d<0:4>` added to it.

Arguments

None

Value Returned

`t`	Successfully displayed Create Patchcord form.
`nil`	Failed to display Create Patchcord form.

Related Topics

Adding Patchcords Using the Create Patchcord Form

schHiCreatePin

```
schHiCreatePin(
     [ ?terminalName t_terminalName ]
     [ ?direction t_direction ]
     [ ?usage t_usage ]
     [ ?interpret t_interpret ]
     [ ?mode t_mode ]
     [ ?netExpr t_netExpr ]
     [ t_justify ]
     [ t_fontStyle ]
     [ n_fontHeight ]
     )
     => t
```

Description

Creates a pin of a specified type in your schematic. Usable only when editing schematics.

Note: See also the `schPinMasters` environment variable which can be used to set the pin masters choices when creating a pin.

Arguments

`?terminalName` *`t_terminalName`*	Terminal name of the pin to create; must be enclosed in quotation marks. To create more than one pin, use a space between names as a delimiter. Each pin is placed individually. If you did not specify *`t_terminalName`* or you specify it as `nil`, the Options form appears.
`?direction` *`t_direction`*	Direction of the pin; must be enclosed in quotation marks. Valid Values: `input`, `output`, `inputOutput`, `switch`
`?usage` *`t_usage`*	Type of pin; must be enclosed in quotation marks. Valid Values: `schematic`, `offSheet`

?interpret *t_interpret*	Interprets terminalName; must be enclosed in quotation marks. If you set *t_interpret* to member, a pin for each member name in *t_terminalName* is generated in the order presented in *t_terminalName*. For example, if you designate a multibit terminal name as addr<7:0>, the schematic editor places each member name, addr<7> through addr<0>, and each of these member name pins individually. If you set *t_interpret* to full, a pin for each space-delimited terminal name from *t_terminalName* is placed individually. Valid Values: full, member Default: full
?mode *t_mode*	Mode you use to place the pins; must be enclosed in quotation marks. If you set *t_mode* to array (placement field), the schematic editor places the current pin as if in single mode. Then, if there are any remaining pins to place, it prompts you for a second point that sets the offset between the remaining pins. When a hierarchical pin exists in a sheet schematic, the schematic editor preserves the direction of the terminal when you create an offsheet connector for the same terminal with a different direction. In this case, the direction specified for the offsheet pin is used only to select its master. Valid Values: single, array Default: single
?netExpr *t_netExpr*	A string containing the net expression in NLP syntax; must be enclosed in quotation marks.

t_justify	Justification to give the label text with respect to its placement; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
t_fontStyle	Label font style; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`
n_fontHeight	Label height in user units. Default: `0.0625`

Value Returned

Always returns `t`.

Examples

Creates a pin for an input offsheet pin. You can drag and place the pin using the mouse. The schematic editor designates the terminal name `data1`.

```
schHiCreatePin( ?terminalName "data1" ?direction "input" ?usage "offSheet" )
```

Adds a net expression to an inout offsheet pin.

```
schHiCreatePin( ?terminalName "data1" ?direction "input" ?usage "offSheet"
?netExpr "[@power:%:vdd!]" )
```

For each member name, `addr<7>` through `addr<0>`, you can drag and place a schematic input pin with a member name.

```
schHiCreatePin( ?terminalName "addr<7:0>" ?direction "input" ?usage "schematic"
?interpret "member" )
```

Places one schematic input pin for each space-delimited name `addr<7:0>`, `data4`, and `data5`.

```
schHiCreatePin( ?terminalName "addr<7:0> data4 data5" ?direction "input" ?usage
"schematic" ?interpret "full" )
```

Places an array of schematic input pins.

```
schHiCreatePin( ?terminalName "addr<2:0>" ?direction "input" ?usage "schematic"
?interpret "member" ?mode "array" )
```

Related Topics

schPinMasters

The Syntax of an Inherited Net Expression

schHiCreateSheet

```
schHiCreateSheet(
     [ ?number n_number ]
     [ ?size t_size ]
     [ ?type t_type ]
     )
     => t
```

Description

Creates a sheet for a multisheet schematic. Usable only when editing schematics.

If you do not supply all arguments, a form appears and prompts you to specify the appropriate values.

If you want to create a multisheet schematic from a nonmultisheet schematic, you are asked to confirm the conversion. The current schematic becomes the first sheet in the index, the created sheet becomes the second sheet, and an index is created. All appropriate properties in the index and on the sheet are updated.

Arguments

?number *n_number*	Number of the sheet to create. If the value is less than zero or specified as `nil`, the schematic editor generates a number based on the value of the last number in the multisheet schematic.
?size *t_size*	Size of the sheet to create; must be enclosed in quotation marks. If `none` is specified, the schematic editor creates a new sheet without a border. If `nil` is specified, the schematic editor designates the sheet size from a property on the index or it defaults to a standard size if the property does not exist. Valid Values: `A`, `B`, `C`, `D`, `E`, `F`, `A Book`, `none`
?type *t_type*	Type of border; must be enclosed in quotation marks. A `continue` sheet border contains less information than the standard sheet border. If `none` is specified for *t_size*, *t_type* is ignored. If no type is specified, a `basic` sheet border is created. Valid Values: `basic`, `continue`

Value Returned

Always returns `t`.

Examples

Creates sheet number three, with a C-sized continuation border, which becomes the current cellview.

```
schHiCreateSheet( 3 "C" "continue" )
```

schHiCreateSymbolLabel

```
schHiCreateSymbolLabel(
     [ ?labelChoice t_labelChoice ]
     [ ?text t_text ]
     [ ?justify t_justify ]
     [ ?fontStyle t_fontStyle ]
     [ ?fontHeight n_fontHeight ]
     [ ?type t_type ]
     )
     => t
```

Description

Places labels in a symbol. Usable only when editing symbols.

Use `schHiCreateSymbolLabel` to create the labels that are normally required in a symbol. For each label to place, the function drags the label specified by *`t_text`* and prompts you for a location to place it. The function clears *`t_text`* and allows setting of various options after each label is placed.

Arguments

`?labelChoice` *`t_labelChoice`*

The kind of label to create; must be enclosed in quotation marks.
Valid Values: `instance label`, `device annotate`, `logical label`, `physical label`, `pin label`, `pin annotate`

`?text` *`t_text`*

Text of the label; must be enclosed in quotation marks.

`?justify` *`t_justify`*

Label justification of the text with respect to its placement; must be enclosed in quotation marks.
Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`

`?fontStyle` *`t_fontStyle`*

Label font style; must be enclosed in quotation marks.
Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`

`?fontHeight` *`n_fontHeight`*

	Label height in user units. Default: `0.0625`
`?type` *`t_type`*	Label type; must be enclosed in quotation marks. Valid Values: `normalLabel`, `NLPLabel`, `ILLabel`

Value Returned

Always returns `t`.

Examples

Drags the instance label `counter` and prompts you for a location to place it.

```
schHiCreateSymbolLabel( "instance label" "counter" )
```

schHiCreateSymbolPin

```
schHiCreateSymbolPin(
     [ ?terminalName t_terminalName ]
     [ ?type t_type ]
     [ ?direction t_direction ]
     [ ?interpret t_interpret ]
     [ ?mode t_mode ]
     [ ?incrCount n_incrCount ]
     [ ?location t_location ]
     )
     => t
```

Description

Creates pins in a symbol. Usable only when editing symbols.

If you do not specify *t_terminalName* or you specify it as `nil`, the options form appears and prompts you to change any of the fields.

While dragging the appropriate pin master to a location, click. If you did not set *t_location* to `none`, a label is placed to display the terminal name.

Arguments

`?terminalName` *t_terminalName*

Terminal name of the pin; must be enclosed in quotation marks. To create more than one terminal name, use a space between names as a delimiter.

`?type` *t_type*

Type of pin; must be enclosed in quotation marks. The value must be an entry in the `schSymbolPinMasters` map.

`?direction` *t_direction*

Direction of the pin; must be enclosed in quotation marks.
Valid Values: `input`, `output`, `inputOutput`, `switch`

`?interpret` *t_interpret*

Interprets *t_terminalName*; must be enclosed in quotation marks. If you set *t_interpret* to member, a pin for each member name in *t_terminalName* is generated in the order presented in *t_terminalName*. If you set *t_interpret* to full, a pin for each space-delimited terminal name from *t_terminalName* is placed individually.
Valid Values: full, member

?mode *t_mode*

Mode used to place pins; must be enclosed in quotation marks. If you set *t_mode* to single, the schematic editor drags each described pin and prompts you for a location to place each one. If you set *t_mode* to array, the schematic editor places the current pin as if in single mode. Then, if there are any remaining pins to place, it prompts you for a second point that sets the offset between the remaining pins.
Valid Values: single, array

?incrCount *n_incrCount*

Distance between the pin and the label.
Valid Values: 0 through 32
Default: 1

?location *t_location*

Location of the terminal name label; must be enclosed in quotation marks.
Value Values: left, right, none

Value Returned

Always returns t.

Examples

Creates a square input symbol pin using the pin master. Drags the pin with a label to the left until you select a destination point. The editor designates the pin name data1.

```
schHiCreateSymbolPin( "data1" "square" "input" "full" "single" 0 "left" )
```

For each member name `addr<7>` through `addr<0>`, you can drag and place a symbol input pin with a label to the left of the pin.

The input pins are named with the individual member names.

```
schHiCreateSymbolPin( "addr<7:0>" "square" "input" "member" "single" 0 "left" )
```

Places a pin for each space-delimited terminal name `data1`, `data2`, and `data3`.

```
schHiCreateSymbolPin( "data1 data2 data3" "square" "input" "full" "single" 0
"left" )
```

Places an array of symbol input pins.

```
schHiCreateSymbolPin( "addr<7:0>" "square" "input" "member" "array" )
```

Related Topics

schSymbolPinMasters

schHiCreateSymbolShape

```
schHiCreateSymbolShape(
     [ ?shape t_shape ]
     [ ?style t_style ]
     [ ?width x_width ]
     [ ?nonModal g_nonModal ]
     )
     => t
```

Description

Creates a line or shapes that describe a symbol. Usable only when editing symbols.

If you did not specify either *t_shape* or *t_style*, the options form appears and prompts you to change the settings. The shapes created with this function give you a visual indication about the symbol's purpose.

Arguments

?shape *t_shape*	Type of shape to create; must be enclosed in quotation marks. Valid Values: `line`, `rectangle`, `polygon`, `arc`, `circle`, `ellipse`
?style *t_style*	Fill style of the shape; must be enclosed in quotation marks. Valid Values: `outline`, `solid`
?width *x_width*	Width of the line, if *t_shape* is set to `line`.
?nonModal *g_nonModal*	Specifies whether the command should be modal. Valid Values: `t`, `nil` Default: `nil`

Value Returned

Always returns `t`.

Examples

Starts creating another rectangle.

```
schHiCreateSymbolShape( "rectangle" "outline" )
```

schHiCreateWire

```
schHiCreateWire(
     [ ?width n_width ]
     [ ?drawMode g_drawMode ]
     [ ?routeMethod t_routeMethod ]
     [ ?lockAngle t_lockAngle ]
     [ ?nonModal g_nonModal ]
     [ ?color t_color ]
     [ ?lineStyle t_lineStyle ]
     )
     => t
```

Description

Creates different style wires that represent net connections in a schematic. Usable only when editing schematics.

You can manually draw a wire or let the schematic editor route the wire automatically, depending on *g_drawMode*. *g_drawMode* allows you to draw wires of different shapes. *t_routeMethod* prompts you to enter two end points for the wire that the schematic editor then routes.

The schematic editor draws a rubberband line until you click the next segment. The current setting for *g_drawMode* determines the rubberband line shape. You can change the value by displaying the options form.

To complete the line manually, click a schematic pin, a pin of a component, another wire, or double-click the wire end point. The schematic editor continually prompts you to click another wire until you want to select another function.

When you complete the wire, the schematic editor automatically generates pins if the edge of the wire connects to the edge of a block. You can specify the pin attributes by setting the *blockDirRule* field of the Editor Options form and by specifying the *pinSeed* property on the block.

Arguments

?width *n_width*	Wire width in user units.
?drawMode *g_drawMode*	Method of wire entry; must be enclosed in quotation marks. When *g_drawMode* is set to anyAngle, the line can be locked to 45 degrees with *t_lockAngle*. Valid Values: route, anyAngle, l90X, l90Y, l45Long, l45Angle

`?routeMethod` *`t_routeMethod`*	Method to use when routing the wires; must be enclosed in quotation marks. Valid only when *`g_drawMode`* is `route`. A *`t_routeMethod`* of `flight` leaves flight lines in the schematic, of `full` executes an algorithm that creates orthogonal line segments, and of `direct` creates a straight solid line between the two points. In some cases, the `full` routing method is not able to complete the connection path. In these cases, a flight line is left in the schematic to indicate the intended connectivity. Valid Values: `flight`, `full`, `direct`
`?lockAngle` *`t_lockAngle`*	Specifies whether the drawing lines are locked to 45 degrees; must be enclosed in quotation marks. Valid only when *`g_drawMode`* is `anyAngle`. Valid Values: `any`, `45`
`?nonModal` *`g_nonModal`*	Specifies whether the command should be modal. Valid Values: `t`, `nil` Default: `nil`
`?color` *`t_color`*	Specifies the color; must be enclosed in quotation marks. The color must be defined in the Display Resource File. If *`t_routeMethod`* is `flight`, *`t_color`* is ignored.
`?lineStyle` *`t_lineStyle`*	Specifies the line style; must be enclosed in quotation marks. The line style must be defined in the Display Resource File. If *`t_routeMethod`* is `flight`, *`t_lineStyle`* is ignored.

Value Returned

Always returns `t`.

Examples

Starts creating a 0.0625-unit wide line in the schematic. The rubberband line shape is set to L90 and starts in the X direction.

```
schHiCreateWire( 0.0625 "l90X" )
```

Prompts you to enter two points for the wire, which the schematic editor then routes using the most complete routing algorithm. Draws a rubberband line from your first selected point until you select a second point. The schematic editor routes the wire automatically using the `full` route method.

```
schHiCreateWire ( 0.0 "route" "full" )
```

schHiCreateWireLabel

```
schHiCreateWireLabel(
     [ ?text t_text ]
     [ ?purpose t_purpose ]
     [ ?justify t_justify ]
     [ ?fontStyle t_fontStyle ]
     [ ?fontHeight n_fontHeight ]
     [ ?interpret t_interpret ]
     [ ?mode t_mode ]
     [ ?netExprFlag g_netExprFlag ]
     [ ?bundleDisplay t_bundleDisplay ]
     )
     => t
```

Description

Creates wire labels in a schematic. Physical contact between a label and wire is not required. You can move a label independently from a wire. When you move a wire that has a label glued to it, the label also moves. Usable only when editing schematics.

Specify `nil` for any option you wish to skip over.

Arguments

`?text` *`t_text`*	Text of the label; must be enclosed in quotation marks. To create more than one label, use a space between labels as a delimiter. The schematic editor places each label individually. If you modify any portion of *`t_text`* while the function is active, label generation begins again from the first label in *`t_text`*.
`?purpose` *`t_purpose`*	Purpose of the placed label; must be enclosed in quotation marks. If you set *`t_purpose`* to `label`, the placed label assigns the given name to the indicated wire by renaming it. If you set *`t_purpose`* to `alias`, the placed label defines an alias for the indicated wire. Valid Values: `label`, `alias`
`?justify` *`t_justify`*	Justification of the label text with respect to its placement location; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`

`?fontStyle` *`t_fontStyle`* Label font style; must be enclosed in quotation marks.
Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`

`?fontHeight` *`n_fontHeight`*

Label height in user units.
Default: `0.0625`

`?interpret` *`t_interpret`* Interpretation of *`t_text`*; must be enclosed in quotation marks. If you set *`t_interpret`* to `member`, a label for each member name listed in *`t_text`* is generated in the order presented in *`t_text`* and each of these member name labels is placed individually. If you set *`t_interpret`* to `full`, a label for each space-delimited terminal name from *`t_text`* is placed individually. When *`t_interpret`* is changed from `member` to `full`, the label text is reset to the full text of the name that generated the current member name.
Valid Values: `full`, `member`

`?mode` *`t_mode`* Mode to place the labels; must be enclosed in quotation marks. If you set *`t_mode`* to `single`, you can drag each label to place it. If you click an open area, a rubberband line is drawn from the control point of the label and prompts you for the location of the wire segment to label. If you set *`t_mode`* to `array`, you are prompted for a location to place the first and second labels. These two points define a directed line that extends from the first point to the second. Any labels remaining to be placed are applied to the wires that cross the directed line. The offset distance from the first wire to the first label is used to offset the remaining labels from the indicated wires at the point that the wires cross the directed line. If the directed line crosses fewer wires than there are remaining labels to place, the function repeats the mode by dragging the next label and prompting for its placement location. If the directed line crosses more wires than there are labels remaining to place, the excess lines are not labeled and the function terminates.
Valid Values: `single`, `array`

`?netExprFlag` *`g_netExprFlag`*

A boolean variable when set to `t` specifies that the *`t_text`* argument can be used as the net expression text. Else, the text is specified through the environment variable `createNetExprText`.

`?bundleDisplay` *`t_bundleDisplay`*

A string indicating the direction of showing wire bundles on the canvas, either `horizontal` or `vertical`.

Value Returned

Always returns `t`.

Examples

Creates individual member names `data<7>` through `data<0>`.

```
schHiCreateWireLabel( "date<7:0>" "label" nil nil nil "member" )
```

Displays the options form and prompts you to specify the label text.

```
schHiCreateWireLabel( nil "label" )
```

Creates individual member names `addr<7>` through `addr<0>` and places the label in the upper left corner of the wire.

```
schHiCreateWireLabel( "addr<7:0>" "label" "member" "upperLeft" )
```

Creates each space-delimited label name: `data1`, `data2`, and `data3`.

```
schHiCreateWireLabel( "data1 data2 data3" "label" "full" "single" )
```

schHiCreateWireStubs

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

Description

Starts the Create Wire Stubs and Names command.

Arguments

None

Value Returned

`t`	Command started.
`nil`	Command failed or returned an error.

Related Topics

Creating Wire Stubs Names

schHiDefaultAction

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

Description

Performs the default editing action for the object under the cursor.

For schematics, this means descending into an instance, editing a label directly on the canvas, or performing `schExtendSelectPt` for any other object. In a symbol, only labels have a default action (on-canvas edit).

Typically, you will want to register this as a mouse bindkey (for example, a double-click of the left mouse button which is the default binding shipped with the schematic editor).

Arguments

None

Value Returned

`t`	The default editing action was successful.
`nil`	The default editing action failed.

Related Topics

schExtendSelectPt

schHiDelete

```
schHiDelete(
     )
     => t
```

Description

Deletes selected objects. Usable when editing schematics or symbols. This function cannot be used to delete a sheet border (use `schHiEditSheetSize`).

If you have not selected the object you want to delete, the editor prompts you to select the object.

Arguments

None

Value Returned

Always returns `t`.

schHiDeleteIndex

```
schHiDeleteIndex(
     )
     => t
```

Description

Converts a multisheet schematic having one drawing sheet and one index sheet into a single-sheet schematic. Usable only when editing an index schematic. Use `schHiDeleteSheet` to first remove extra drawing sheets. The index schematic must contain only one sheet.

The index is deleted and the single remaining sheet is converted into a non-multisheet schematic. Any offsheet connectors in the design are converted to terminals of the same direction. The design name is preserved. The sheet symbol is deleted. If a border exists in the converted schematic, it is replaced with its single-sheet equivalent.

Arguments

None

Value Returned

Always returns `t`.

schHiDeleteSheet

```
schHiDeleteSheet(
     [ ?startSheet n_startSheet ]
     [ ?endSheet n_endSheet ]
     )
     => t
```

Description

Deletes a sheet or range of sheets from a multisheet schematic. Usable only when editing an index of a multisheet schematic. This function cannot be undone.

If you do not specify a start sheet number, the options form appears. If the end sheet value is equal to the start sheet, only one sheet is deleted. You are prompted to confirm the deletion before proceeding because the action cannot be undone.

Arguments

`?startSheet` *`n_startSheet`*	First sheet number to delete. If you do not specify the end sheet, only the current sheet is deleted.
`?endSheet` *`n_endSheet`*	Last sheet number to delete.

Value Returned

Always returns `t`.

Examples

Deletes the fourth sheet. You are prompted to confirm the deletion.

```
schHiDeleteSheet( 4 )
```

Deletes the third, fourth, fifth, and sixth sheets. You are prompted to confirm the deletion.

```
schHiDeleteSheet( 3 6 )
```

schHiDescend

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

Description

Traverses down the hierarchy and displays the child cellview of a specified instance and view you select, provided you have edit permission. If you do not have edit permission, a dialog box prompts you to use read mode.

The view you descend into is displayed in either the current window or in a new window, depending on whether you have enabled the *Create New Window When Descending* option on the User Preferences form. You access the User Preferences form from the *Options – User Preferences* command on the CIW.

If the selected instance represents a multisheet or index schematic, a form appears. You use the form to specify the number of the sheet you want to descend into.

Arguments

w_windowId	A window ID indicating the cellview from which you want to start the descend command. If not specified, the current window is used.

Value Returned

Always returns `t`.

schHiDescendEdit

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

Description

Traverses down the hierarchy and displays the child cellview of a specified instance and view you select, provided you have edit permission. If you do not have edit permission, a dialog box prompts you to use read mode.

The view you descend into is displayed in either the current window or in a new window, depending on whether you have enabled the *Create New Window When Descending* option on the User Preferences form. You access the User Preferences form from the *Options – User Preferences* command on the CIW.

If the selected instance represents a multisheet or index schematic, a form appears. You use the form to specify the number of the sheet you want to descend into.

Arguments

w_windowId	The window ID indicating the cellview from which you want to start the descend command. If not specified, the current window is used.

Value Returned

Always returns `t`.

schHiDescendRead

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

Description

Traverses down the hierarchy and displays the child cellview in read mode of a specified instance and view you select. The cellview is not editable.

The view you descend into is displayed in either the current window or in a new window, depending on whether you have enabled the *Create New Window When Descending* option on the User Preferences form. You access the User Preferences form from the *Options – User Preferences* command on the CIW.

If the selected instance represents a multisheet or index schematic, a form appears. You use the form to specify the number of the sheet you want to descend into.

Arguments

w_windowId	A window ID indicating the cellview from which you want to start the descend command. If not specified, the current window is used.

Value Returned

Always returns `t`.

schHiDisplayOptions

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

Description

Sets options associated with the window display. Usable when editing schematics or symbols.

Opens a form where you can set display options.

Arguments

w_windowId	Window to be modified. If not specified, the current window is modified.

Value Returned

Always returns `t`.

schHiDistribute

```
schHiDistribute(
     s_justify
     [ g_windowId ]
     )
     => t
```

Description

Arranges the objects on the canvas at equal distance from each other. If the objects are already selected on the canvas, the pre-select mode is used; otherwise, the post-select mode is used for distributing the objects.

Arguments

s_justify	The direction in which the objects need to be distributed. Possible values are `vertical` and `horizontal`.
g_windowId	The window indicating the cellview where you want to start distributing the objects. If not specified, the current window is used. This is an optional argument.

Value Returned

`t`	Objects were distributed successfully.
`nil`	Objects could not be arranged.

Examples

```
schHiDistribute( 'vertical )
schHiDistribute( 'horizontal hiGetCurrentWindow() )
schHiDistribute( 'vertical window(3) )
```

Related Topics

Distributing

schHiDrawSymbolPin

```
schHiDrawSymbolPin(
     [ ?terminalName t_terminalName ]
     [ ?direction t_direction ]
     [ ?interpret t_interpret ]
     )
     => t
```

Description

Draws a polygon that represents a symbol pin. Usable only when editing symbols.

Arguments

?terminalName *t_terminalName*

	Terminal name of the pin being drawn; must be enclosed in quotation marks. To create more than one pin, use a space between names as a delimiter. Each pin is placed individually.
?direction *t_direction*	Direction of the pin; must be enclosed in quotation marks. Value values: `input`, `output`, `inputOutput`, `switch`
?interpret *t_interpret*	Interprets *t_terminalName*; must be enclosed in quotation marks. Valid Values: `full` (full name), `member`

Value Returned

Always returns `t`.

Examples

Prompts you to draw a polygon to represent a pin. Direction of the pin is `input`. The `data1` terminal name is not displayed with the pin.

```
schHiDrawSymbolPin( data1 "input" )
```

Prompts you to draw a polygon to represent a pin. Direction is the current or last value. The `data2` terminal name is not displayed with the pin.

```
schHiDrawSymbolPin( "data2" )
```

schHiEditInPlace

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

Description

Edits the master of a user-selected instance within the context of its parent schematic. Usable when editing schematics or symbols.

If an instance is on the selected set, the instance is edited in place. Otherwise the Virtuoso® Schematic Editor prompts you to select an instance. If you do not have edit access to the master of the instance, a warning message is displayed.

Arguments

w_windowId	A window indicating the cellview where you want to start the command. If not specified, the current window is used.

Value Returned

Always returns `t`.

Related Topics

schHiReturn

schHiEditText

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

Description

Starts the direct text edit command in the current window, allowing you to perform an in-place edit of notes, names, parameters, and so on.

Arguments

None

Value Returned

`t`	Action a success.
`nil`	Action failed

schHiEditorOptions

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

Description

Sets variables that affect the environment. Usable when editing schematics or symbols.

Displays a form that you use to modify options in the editing environment.

Arguments

w_windowId	Window to be modified. If not specified, the current window is modified.

Value Returned

Always returns `t`.

schHiEditPinOrder

```
schHiEditPinOrder(
     [ g_updateInstLastChanged ]
     )
     => t
```

Description

Creates or modifies the property on the cellview that specifies the ordering of the pins in the current cellview. Usable when editing schematics or symbols.

Important

This function replaces `schHiUpdatePinOrder`. You should use `schHiEditPinOrder` instead of `schHiUpdatePinOrder`.

Operates on the current cellview. It creates or modifies a property on the cellview that describes each pin in the cellview. The importance of the property is to maintain a specific ordering of the pins that can be synchronized against the port ordering of an HDL instance.

Arguments

g_updateInstLastChanged

Specifies whether the time stamp on the cellview is modified.

Value Returned

Always returns `t`.

schHiEditSheetSize

```
schHiEditSheetSize(
     [ ?size t_size ]
     [ ?type t_type ]
     )
     => t
```

Description

Edits the sheet border size of a schematic. Usable only when editing schematics.

If you edit a schematic that contains a sheet border, the function changes the border size and type specified. A warning is displayed in a dialog box if data is outside the new border.

Arguments

`?size` *`t_size`*	Specifies a new sheet border size; must be enclosed in quotation marks. If you specify `none`, any existing border is deleted from the schematic. Valid Values: `A`, `B`, `C`, `D`, `E`, `F`, `A Book`, `none`
`?type` *`t_type`*	Specifies the sheet type; must be enclosed in quotation marks. This argument is ignored if you set *`t_size`* to `none`. Valid Values: `basic`, `continue`, `single` Default: `basic`

Value Returned

Always returns `t`.

Examples

Displays the options form with sheet size set to `C`.

```
schHiEditSheetSize( "C" )
```

Changes the current sheet size to `D` and the border to `continue`.

```
schHiEditSheetSize( "D" "continue" )
```

schHiEditTitleBlock

```
schHiEditTitleBlock(
     )
     => t
```

Description

Changes the properties of a title block in a schematic sheet. Usable only when editing schematics that have a sheet border.

One of two forms appears with this function:

- The first form appears if you call this function when you edit the index of a multisheet schematic. This form lists the title block properties that apply across all sheets in a multisheet schematic.
- The second form appears if you call this function when you edit a sheet. This form lists those title block properties that are specific to a given sheet.

If you call this function from a non-multisheet schematic that contains a border, a combination form appears where all properties applicable to the title border are presented for editing.

Arguments

None

Value Returned

Always returns `t`.

schHiEnvSaveLoad

```
schHiEnvSaveLoad(
     [ ?action t_action ]
     [ ?fileName t_fileName ]
     )
     => t
```

Description

Saves or loads the schematic environment variables. Usable when editing schematics or symbols.

If *t_action* is `save`, the current environment changes are saved to the named file. If *t_action* is `load`, the environment variables specified in *t_fileName* are read in and any corresponding form defaults are set. This is useful if you saved defaults to a file other than `~/.cdsenv`.

Arguments

t_action	Specifies the action this function should take; must be enclosed in quotation marks. Valid Values: `save`, `load`
t_fileName	The name of the file from which the schematic environment variables will be saved to or loaded from; must be enclosed in quotation marks. If no filename is specified, the variables are saved in `~/.cdsenv`.

Value Returned

Always returns `t`.

Examples

Saves the schematic environment variables and their current values to the `/tmp/schenv` file.

```
schHiEnvSaveLoad( "save" "/tmp/schenv" )
```

schHiEscape

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

Description

Closes an open form or clears the current canvas selection. You would typically use this function to successively close the open forms on your desktop, even if they are hidden behind other windows. If there is only one form open, it is closed. If there are multiple forms open, the first form returned by the `hiGetCurrentForm` SKILL API is closed. If there are no forms open, the current canvas selection is cleared.

Arguments

None

Value Returned

`t`	Form closed or selection cleared successfully.
`nil`	No action done.

Examples

Closes a form or clears the current selection in the canvas.

```
schHiEscape()
```

schHiExtractConn

```
schHiExtractConn(
     [ ?action t_action ]
     )
     => t
```

Description

Sets extraction options and runs the schematic extractor. Usable only when editing schematics.

If you do not specify an action, or specify it as `editOptions`, the form appears for you to modify the settings of the various schematic rule checks. If you specify `run`, the connectivity is extracted from the current schematic. If you specify `editOptionsAndRun`, the form appears so that you can change the connectivity options.

The total number of errors and warnings detected is displayed in a dialog box. You must correct detected errors before the connectivity in the schematic is marked as valid.

Arguments

`?action` *t_action*	Edits extraction options, runs the schematic extractor, or both; must be enclosed in quotation marks. Valid Values: `editOptions`, `run`, `editOptionsAndRun`

Value Returned

Always returns `t`.

Examples

Extracts the connectivity from the current schematic.

```
schHiExtractConn( "run" )
```

schHiFind

```
schHiFind(
     [ ?propName t_propName ]
     [ ?condOp t_condOp ]
     [ ?propValue t_propValue ]
     )
     => t
```

Description

Finds objects that match specified search criteria in a schematic or symbol view. You can specify the object filter as well as a property name or value expression that matches the objects. Usable when editing schematics or symbols. Matching is supported only for strings, integers, floating-point numbers, and time. Other property types are not supported.

The `schHiFind` function searches through the schematic or symbol cellview for objects that match the *t_propName*, *t_condOp*, and *t_propValue* arguments. It highlights the first object that matches the search criteria and opens the Schematic Find form.

If *t_propName* is `master`, *t_propValue* must be *libName cellName viewName* (separated by spaces). Wildcards are not supported for the `master` property.

Note: See also the `schVerboseFind` environment variable which can be used to control the messages output in the CIW and `CDS.log` file when *find* criteria has not been met.

Arguments

`?propName` *t_propName*	The property name used in the search criteria; must be enclosed in quotation marks.
`?condOp` *t_condOp*	The conditional operator that is applied to the property name and property value for matching; must be enclosed in quotation marks. Valid Values: `==`, `!=`, `<=`, `>=`, `<`, `>`
`?propValue` *t_propValue*	The value of the property; must be enclosed in quotation marks. The value may include wildcard expressions.

Value Returned

Always returns `t`.

Examples

Displays the options form with *`t_propName`* set to `name`, *`t_condOp`* set to `==`, and *`t_propValue`* set to `I1`. All objects that have `name` set to `I1` are added to the list. The first one is highlighted on the screen.

```
schHiFind( "name" "==" "I1" )
```

Related Topics

schVerboseFind

schHiFindMarker

```
schHiFindMarker(
     )
     => t
```

Description

Searches for the error and warning markers generated by the schematic rules checker (SRC) and displays a form that contains a list of the markers. Usable when editing a schematic or a symbol.

You invoke `schHiFindMarker` after you check your design. A form appears that contains a list of markers, if there are errors. When you click a marker in the list, the system highlights the corresponding marker in the design window and, optionally, zooms in so you can edit the design.

The form also contains options that let you control which markers the system displays.

Arguments

None

Value Returned

Always returns `t`.

schHiFlatten

```
schHiFlatten(
     )
     => t
```

Description

Starts the schematic *Flatten* command. Flattening brings the contents of the flattened instance up to the current level to allow editing while preserving the connectivity of the flattened instance. If no single, valid instance is pre-selected, it prompts you to select an instance to flatten. When a valid instance is selected, it prompts you to select a point to place the flattened content. In the case of schematic-in-schematic, the flattened content is placed immediately, without prompting for an additional point.

Arguments

None

Value Returned

`t`	Flatten successfully completed.
`nil`	Flatten failed.

schHiFollowPin

```
schHiFollowPin(
     [ ?order t_order ]
     )
     => t
```

Description

Changes the cellview to one that contains the specified pin or offsheet connector. Operates on the selected set. Brings in the cellview that contains the first reference of the selected pin or offsheet connector. Usable only when editing a sheet of a multisheet schematic.

You must first compute the pin references using the `schHiComputePinRef` function before you can use this function.

If you have not selected the pin or offsheet connector, the schematic editor prompts you to select the pin or offsheet connector.

The schematic editor zooms into the area that contains the specified pin or offsheet connector. For example, if you selected `first`, the schematic editor displays the cellview that contains the first reference of the pin or offsheet connector you selected.

Use the `schHiComputePinRef` function to generate the reference lists for the pins in the multisheet schematic.

Arguments

`?order` *t_order*	Specifies the relative location of other pins across the sheets in a multisheet schematic that reference the same net; must be enclosed in quotation marks. Valid Values: `first`, `last`, `next`, `previous`

Value Returned

Always returns `t`.

Examples

Changes the cellview to the first view that contains the specified pin or offsheet connector.

```
schHiFollowPin ( "first" )
```

Related Topics

schHiComputePinRef

schHiFontUpdate

```
schHiFontUpdate(
     [?increase x_increase]
     [?amount x_amount]
     [x_increase [x_amount] ]
     )
     => t / nil
```

Description

Interactive command for updating the font size of text on a schematic.

Arguments

`?increase` *x_increase*	Sets that the font size is to be increased.
`?amount` *x_amount*	Number of user units to increase font size on schematic by.

Value Returned

`t`	Font size successfully increased.
`nil`	Command failed.

Examples

Increases font size by two user units.

```
schHiFontUpdate(?increase t ?amount 2)
```

schHiGotoSheet

```
schHiGotoSheet(
     [ ?sheetNum g_sheetNum ]
     )
     => t
```

Description

Traverses sheets in a multisheet schematic. Usable only when editing an index or sheet of a multisheet schematic.

If you do not specify a sheet number, the form appears for you to indicate a value.

Arguments

`?sheetNum` *`g_sheetNum`*	An integer that indicates a sheet number or a string that describes the sheet number by the keywords `first`, `last`, `previous`, `next`, `index`.

Value Returned

Always returns `t`.

Examples

Views or edits sheet number 3 in the current window.

```
schHiGotoSheet( 3 )
```

schHiGridOptions

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

Description

Sets options associated with the grid.

Displays a form, which you use to set the options *drawaxes*, *gridtype*, *gridspacing*, *gridmultiple*, and *snapspacing*.

Arguments

w_windowId	Window to be modified. If not specified, the current window is modified.

Value Returned

Always returns `t`.

schHiHiliteLabel

```
schHiHiliteLabel(
     [ ?labelType t_labelType ]
     [ ?display t_display ]
     )
     => t
```

Description

Highlights labels of wires and instances. Usable only when editing schematics.

The current set of highlighted labels is automatically unhighlighted before highlighting another label type. Highlights or unhighlights the objects that match *t_labelType* when *t_display* is `on` or `off`.

Arguments

`?labelType` *t_labelType*	The type of labels to be highlighted; must be enclosed in quotation marks. Valid Values: `wire`, `instance`, `both`
`?display` *t_display*	Defines if the highlight is to be turned on or off; must be enclosed in quotation marks. Valid Values: `on`, `off`

Value Returned

Always returns `t`.

Examples

Highlights all wire labels and the wires to which the labels are glued.

```
schHiHiliteLabel( "wire" "on" )
```

schHiIgnore

```
schHiIgnore(
     [ g_windowId ]
     )
     => t / nil
```

Description

Ignores the instances in the schematic canvas. This function works as a toggle switch, that is, it is used for adding ignore properties to an instance or for removing any existing ones. When the instances are already selected on the canvas, the pre-select mode is used; otherwise, the post-select mode is used for ignoring the instances.

Arguments

g_windowId	The window indicating the cellview in which you want to ignore the instances. If not specified, the current window is used. This is an optional argument.

Value Returned

`t`	Addition or removal of ignore properties was successful.
`nil`	Operation was unsuccessful.

Examples

```
schHiIgnore()
schHiIgnore( ?windowId hiGetCurrentWindow() )
schHiIgnore( ?windowId window(3) )
```

Related Topics

Ignoring Instances

schHiInstToView

```
schHiInstToView(
     [ ?viewName t_viewName ]
     [ ?dataType t_dataType ]
     )
     => t
```

Description

Generates a cellview from an instance of a symbol. Usable only when editing schematics.

If you have not selected an instance of a symbol in your current schematic, the schematic editor prompts you to select an instance.

If you do not specify a value for any argument or if you specify an argument as `nil`, a form appears to prompt you to specify the field values.

If the `schViewToPinListReg` list or the `schPinListToViewReg` list does not contain the desired view type and conversion function, you must modify the lists. The lists are defined in your `schConfig.il` file (*your_install_dir*`/samples/local/schConfig.il`). Refer to the following section in your `schConfig.il` file for more information about modifying the `schViewToPinListReg` and `schPinListToViewReg` lists: `REGISTERING CONVERSION FUNCTIONS FOR THE CREATE CELLVIEW FROM CELLVIEW COMMANDS.`

Arguments

`?viewName` *t_viewName*	View name of the cellview to be generated; must be enclosed in quotation marks. Valid Values: `symbol`, `schematic`, `functional`, `behavioral`, `system`
`?dataType` *t_dataType*	String corresponding to an entry in the `schPinListToViewReg` list that specifies how the created cellview will be generated; must be enclosed in quotation marks.

Value Returned

Always returns `t`.

Examples

Generates a schematic view.

```
schHiInstToView ( "schematic" "Schematic" )
```

schHiMakeCell

```
schHiMakeCell(
     [ ?cutArea n_cutArea ]
     )
     => t
```

Description

Starts the schematic *Make Cell* command. *Make Cell* cuts out a rectangular region of the schematic, moving it into a new schematic cellview, and fills in the gap with a new hierarchical instance. The command adjusts the design hierarchy without changing the connectivity. You can define the rectangular area you wish to cut.

Arguments

`?cutArea` *`n_cutArea`*

Specifies the bounding box, in the standard SKILL format, to be used as the initial area from which to make a new cell. If you do not specify the cut area, the *Make Cell* form opens where you can provide the required information before proceeding.

Value Returned

Always returns `t`.

Examples

Starts the *Make Cell* command in the current window, and prompts you to define an initial cut area before displaying the Make Cell form.

```
schHiMakeCell
```

Starts the *Make Cell* command in the current window and immediately displays the Make Cell form for cutting out the rectangle defined by the bounding box coordinates (5, 20) - (10, 25).

```
schHiMakeCell ?cutArea '((5 20) (10 25))
```

schHiMousePopUp

```
schHiMousePopUp(
     )
     => t
```

Description

Displays a pop-up menu at the location of the cursor. The menu displayed is sensitive to the location of the cursor and the mode of the cellview in the window. Usable when editing schematics or symbols.

Selection is dependent on the `schHiSetSelectOptions` settings.

For schematic and symbol views, a set of menu categories have been predefined. Each category defines the conditions in which the cursor will display that menu. In addition, each category has a menu that you can customize and to which you can assign `read` and `edit` access modes.

When only one object is in the selected set, the type of object determines the category. Context sensitivity can be turned off with the *sensitiveMenu* option. When turned off, the `schStandard` and `symStandard` categories are used.

Category	Description
instance	Schematic instance
index sheet	Sheet instance in an index schematic
border	Sheet border in a schematic cellview
schPin	Schematic pin
symPin	Symbol pin
indexPin	Pin in an index schematic
instPin	Schematic instance pin
indexInstPin	Instance pin in an index schematic
wire	Schematic wire, path, or flightline
label	Label (note, symbol, or wire labels)
shapes	Shapes (note or device shapes)
marker	Markers

Category	Description
indexDefault	Any other object in an index schematic
schDefault	Unknown object in a schematic cellview
symDefault	Unknown object in a symbol cellview
schematic	No object under cursor in a schematic cellview
symbol	No object under cursor in a symbol cellview
symSelSet	Multiple objects selected in a symbol cellview
schSelSet	Multiple objects selected in a schematic cellview
schStandard	Schematic menu to be used when context sensitivity is off
symStandard	Symbol menu to be used when context sensitivity is off

Arguments

None

Value Returned

Always returns `t`.

Related Topics

schHiDisplayOptions

schHiMove

```
schHiMove(
     [ ?useSelSet t_useSelSet ]
     )
     => t
```

Description

Moves objects to a different location. You can move objects between different schematic and symbol cellviews. Usable when editing schematics or symbols. Partially selected objects cannot be moved to a different cellview. Does not support change in orientation for partially selected objects.

The argument *t_useSelSet* determines whether the function operates on the selected set. If *t_useSelSet* is `useSelSet` and the selected set contains at least one object, you are prompted to click a reference point. If the selected set is empty or *t_useSelSet* is `noSelSet`, you are prompted to point at an object to move; that coordinate is also the reference point. Also, if *t_useSelSet* is `noSelSet`, the function is nonmodal.

When the point is specified, all connectivity is broken between the selected objects and the objects they are connected to. New connectivity is computed when the objects are placed at their destination.

Arguments

t_useSelSet	Specifies whether to move the selected set or an object you choose with the mouse; must be enclosed in quotation marks. Valid Values: `useSelSet`, `noSelSet` Default: `useSelSet`

Value Returned

Always returns `t`.

Examples

Prompts you to select objects to move, ignoring the current selected set.

```
schHiMove( "noSelSet" )
```

schHiNetExprAvailProps

```
schHiNetExprAvailProps(
     )
     => t
```

Description

Displays the Net Expression Available Property Names form. Use this form to find the net expression property names that are available for setting on the selected instances.

Arguments

None

Value Returned

Always returns `t`.

schHiNetExprEvalNames

```
schHiNetExprEvalNames(
     )
     => t
```

Description

Displays the Net Expression Evaluated Names form. Use this form to view the net names that result from the evaluation of all net expressions that are found in the hierarchy below the selected instances.

Arguments

None

Value Returned

Always returns `t`.

schHiNewCellView

```
schHiNewCellView(
     )
     => t
```

Description

Displays the Create New File form. This function needs to be invoked from a window that contains either schematic or symbol data.

You specify the library name, the new cell name, the new view name, and the tool (either *Schematic Editor – Schematic* or *Schematic Editor – Symbol*).

Arguments

None

Value Returned

Always returns `t`.

schHiObjectProperty

```
schHiObjectProperty(
     )
     => t
```

Description

Displays a form that lets you modify the properties of selected objects. Usable only when editing a schematic or symbol cellview.

Operates on the selected set. If the selected set is empty or contains no suitable object types, the form is mostly empty. If there are multiple objects in the selected set, you can step through each of them either sequentially or by object type.

The properties for the current object are displayed in a form. The contents of the form are automatically updated when you select another object.

You can modify multiple objects of the same type simultaneously. You can also control the visibility of some properties.

Arguments

None

Value Returned

Always returns `t`.

schHiOpenCellView

```
schHiOpenCellView(
     )
     => t
```

Description

Displays the Open File form. You specify the library name, cell name, view name, and edit mode in this form. The specified cellview is displayed in the current window. Usable for opening schematic and symbol design windows.

Arguments

None

Value Returned

Always returns `t`.

schHiOpenOtherView

```
schHiOpenOtherView(
     d_cellViewID
     [ t_viewNames ]
     [ w_windowID ]
     [ w_sessionWindowID ]
     )
     => nil / windowID
```

Description

Allows you to switch between schematic and symbol views.

If `t_viewNames` only has one entry, then `schHiOpenOtherView` will open that view. However, if `t_viewNames` is not specified, or = nil, then the command will determine what views exist (excluding the current cellview). If any views do exist, a dialog box will be displayed prompting you to select a view to open. If no views exist, a warning message will be displayed informing you that there are no views that can be opened.

Consider the following relationships between `w_windowID` and `w_sessionWindowID`:

- If both `w_windowID` and `w_sessionWindowID` are nil, a new session window will be opened to display the cellview
- If `w_sessionWindowID` is set to an existing session window, and `w_windowID` is nil, then a new tab will be opened in the existing session window to display the cellview.
- If `w_sessionWindowID` is nil, and `w_windowID` is set to an existing window, then the existing window will be reused to display the cellview.
- If `w_sessionWindowID` is set to an existing session window, and `w_windowID` is set to an existing tab, then that tab will be used to display the cellview if it belongs to the specified session window. If however the tab corresponding to the `w_windowID` belongs to a different session window than the window specified in `w_sessionWindowID`, then an error will be displayed.

Note: The UI menu equivalents are *File – Open Schematic*/*Symbol*.

Arguments

d_cellViewID	Specifies the view type that you want to update the current view to, for example “`schematic`”.
t_viewNames	Lists the view names(s) that you can choose to open. These must be enclosed in quotation marks.

w_windowID	Specifies the window/tab to be used to display the newly opened view.
w_sessionWindowID	Specifies the session window to display the opened cellview in.

Value Returned

nil	Indicates that the function has failed. For example, the view may not exist, or an attempt has been made to switch to schematic/symbol view from a layout view, or if the operation is canceled when asked to select a view.
windowID	Displays which window was updated, or created, with a new view.

Examples

Raises a dialog box asking you to select an alternative view which, once selected, will be opened in a *new* window. If there is only one possible view to open, this view will be opened automatically in the new window.

```
schHiOpenOtherView(geGetEditCellView())
```

Raises a dialog box asking you to select an alternative view which, once selected, will be opened in the *current* window. If there is only one possible view to open, this view will be opened automatically in the current window.

```
schHiOpenOtherView(geGetEditCellView() nil hiGetCurrentWindow())
```

Offers the choice of three views (schematic, symbol, and layout), and will open the selected view, if it exists, in a new window.

```
schHiOpenOtherView(geGetEditCellView() list("schematic" "symbol" "layout"))
```

schHiOpenSymbolOrSchematicView

```
schHiOpenSymbolOrSchematicView(
     d_cellViewID
     [ w_windowID ]
     )
     => nil / windowID
```

Description

Takes the current viewtype, schematic or symbol, then calls schHiOpenOtherView to then switch between the views.

Note: The UI menu equivalent is *File – Open Schematic/Symbol*.

Arguments

d_cellViewID	Specifies the view type that you want to update the current view to, for example "`symbol`".
w_windowID	Specifiies the window to be used to display the newly opened view. If no window is specified, the current window will be used.

Value Returned

`nil`	Indicates that the function has failed. For example, the view may not exist, or an attempt has been made to switch to schematic/symbol view from a layout view, or if the operation is canceled when asked to select a view.
`windowID`	Displays which window was updated, or created, with a new view.

Examples

Switches to the schematic view, opening it in the *current* window, if the symbol view is currently open, and vice versa. This will perform the same action as *File – Open Schematic/Symbol* or using the bindkey "`Ctrl =`".

```
schHiOpenSymbolOrSchematicView(geGetEditCellView() hiGetCurrentWindow())
```

As above, but will open the view in a *new* window.

```
schHiOpenSymbolOrSchematicView(geGetEditCellView())
```

Related Topics

schHiOpenOtherView

schHiPinListToView

```
schHiPinListToView(
     [ ?libName t_libName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     [ ?inPinList t_inPinList ]
     [ ?outPinList t_outPinList ]
     [ ?ioPinList t_ioPinList ]
     [ ?swPinList t_swPinList ]
     [ ?dataType t_dataType ]
     [ ?jpPinList t_jpPinList ]
     [ ?trPinList t_trPinList ]
     [ ?unPinList t_unPinList ]
     )
     => t
```

Description

Generates a cellview from a pin list. Usable when editing schematics or symbols.

If you do not specify *t_libName*, *t_cellName*, or *t_viewName*, a form appears for you to specify them. The form also appears if you do not specify all four of the pin list arguments. Use an empty string (`""`) to represent an empty pin list.

If you do not specify any argument or specify an argument as `nil`, the form appears so that you can specify the field values.

If the `schViewToPinListReg` list or the `schPinListToViewReg` list does not contain the desired view type and conversion function, you must modify the lists. The lists are defined in your `schConfig.il` file (*your_install_dir*`/samples/local/schConfig.il`). Refer to the following section in your `schConfig.il` file for more information about modifying the `schViewToPinListReg` and `schPinListToViewReg` lists.

Arguments

`?libName` *t_libName*	Library name to which to append the generated cellview; must be enclosed in quotation marks.
`?cellName` *t_cellName*	Cell name for the generated cellview; must be enclosed in quotation marks.

`?viewName` *`t_viewName`*	View name of the cellview to be generated; must be enclosed in quotation marks. Valid Values: `symbol`, `schematic`, `functional`, `behavioral`, `system`
`?inPinList` *`t_inPinList`*	The input pin names separated by spaces; must be enclosed in quotation marks.
`?outPinList` *`t_outPinList`*	The output pin names separated by spaces; must be enclosed in quotation marks.
`?ioPinList` *`t_ioPinList`*	The input/output pin names separated by spaces; must be enclosed in quotation marks.
`?swPinList` *`t_swPinList`*	The switch pin names separated by spaces; must be enclosed in quotation marks.
`?dataType` *`t_dataType`*	String corresponding to an entry in the `schPinListToViewReg` list that specifies how the created cellview will be generated; must be enclosed in quotation marks.
`?jpPinList` *`t_jpPinList`*	String value of the *Jumper Pins* field.
`?trPinList` *`t_trPinList`*	String value of the *Tristate Pins* field.
`?unPinList` *`t_unPinList`*	String value of the *Unused Pins* field.

Value Returned

Always returns `t`.

Examples

Displays the options form for creating a symbol cellview with input terminals `a` and `b` and output terminal `x`.

```
schHiPinListToView( "mylib" "and2" "symbol" "a" "b" "x" "" "Schematic Editor-
Symbol" )
```

schHiPlot

```
schHiPlot(
     )
     => t
```

Description

Brings up the Submit Plot form to let you generate schematic plots. Usable when viewing schematics or symbols.

Reads the `.cdsplotinit` file.

In the Submit Plot form, you can specify which cellview to plot, which sheet to plot (for multisheet schematics), hierarchical plotting scope, number of copies to produce, the time to start the plot job, a filename if you want to send the plot to an output file instead of a printer, and a plot template file to store or save the options.

Arguments

None

Value Returned

Always returns `t`.

Related Topics

schPlot

schHiPlotQueueStatus

```
schHiPlotQueueStatus(
     )
     => t
```

Description

Displays the plot jobs in the spooling queues. Usable when editing schematics or symbols.

You can use this function to examine the plot spooling queue. You can delete plot jobs that you own.

Arguments

None

Value Returned

Always returns `t`.

schHiRaiseCurrentForm

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

Description

Brings the most recently-used open form to the front. If there are no open forms, the command does nothing. Bound to `F3` by default.

Arguments

None

Value Returned

`t`	The most-recently used form is brought to the front.
`nil`	No forms are open.

schHiRegisterWireStubs

```
schHiRegisterWireStubs(
     )
     => t
```

Description

Invokes the Register Net Name per Terminal form.

Arguments

None

Value Returned

Always returns `t`.

Related Topics

Registering Default Net Names

schHiRenumberAllSheet

```
schHiRenumberAllSheet(
     )
     => t
```

Description

Sequentially renumbers all sheets in a multisheet schematic. Usable only when editing an index schematic.

Numbering starts with `1` and increments by one for every sheet. This function does not change sheet borders. The values displayed in the title block change to reflect any changed sheet numbers.

Arguments

None

Value Returned

Always returns `t`.

schHiRenumberInstances

```
schHiRenumberInstances(
     )
     => t
```

Description

Opens the Renumber Instances form which can be used to renumber instances automatically.

Arguments

None

Value Returned

Always returns `t`.

schHiRenumberSheet

```
schHiRenumberSheet(
     [ ?from n_from ]
     [ ?to n_to ]
     )
     => t
```

Description

Changes the number of a sheet in a multisheet schematic. Changes the cell name of the renumbered schematic to match the destination sheet number. If a sheet already exists with the destination number, the renumbered sheet is inserted before it. All succeeding sheets will be renumbered accordingly.

Arguments

`?from` *`n_from`*	The number of the sheet to renumber.
`?to` *`n_to`*	The new number for the sheet.

Value Returned

Always returns `t`.

Examples

Renumbers sheet number 1 to sheet number 2. If sheet number 2 already exists, you are prompted to continue. If you want to continue, sheet number 2 becomes sheet number 3 and all succeeding sheets are renumbered accordingly.

```
schHiRenumberSheet( 1 2 )
```

schHiReplace

```
schHiReplace(
     [ ?replaceAll g_replaceAll ]
     [ ?propName t_propName ]
     [ ?condOp t_condOp ]
     [ ?propValue t_propValue ]
     [ ?newPropName t_newPropName ]
     [ ?newPropValue t_newPropValue ]
     )
     => t
```

Description

Replaces objects that match the specified search criteria with a specified value. Usable when editing symbols or schematics. You must have write access to each of the cellviews. Only supports matching of strings, integers, floating-point numbers, and time. This function does not support other property types.

The search criteria let you, among other tasks,

- Specify the object filter and a property name or a value expression that the objects must match
- Replace all objects at one time or view each matching item in turn and either replace the item or skip to the next item
- Search in another library or in another cellview and change a property

The `schHiReplace` function searches through the schematic or symbol cellview for objects that match the *t_propName*, *t_condOp*, and *t_propValue* arguments.

Arguments

`?replaceAll` *g_replaceAll*

Specifies whether the replacement is done automatically or interactively. If *g_replaceAll* is set to `t`, the replacement is done automatically.

`?propName` *t_propName*

The name of the property used in the search criteria; must be enclosed in quotation marks. If *t_propName* is set to `master`, *t_propValue* must be *t_libName t_cellName t_viewName* (separated by spaces). No wildcards are supported for `master` property.

`?condOp` *`t_condOp`*	The conditional operator applied to the property name and property value for matching; must be enclosed in quotation marks. Valid Values: `==`, `!=`, `<=`, `>=`, `<`, `>`
`?propValue` *`t_propValue`*	The value of the property; must be enclosed in quotation marks. The value may include wildcard expressions.
`?newPropName` *`t_newPropName`*	The name of the property to replace on each of the matching objects; must be enclosed in quotation marks.
`?newPropValue` *`t_newPropValue`*	The value of the new property assigned to each of the matching objects; must be enclosed in quotation marks.

Value Returned

Always returns `t`.

Examples

Assigns all objects with the name of `I1` to `I2`.

```
schHiReplace( t "instName" "==" "I1" "instName" "I2" )
```

Adds a new property name `newProp` with value set to `I2` to all objects with the name `I1`.

```
schHiReplace( t "instName" "==" "I1" "newProp" "I2" )
```

schHiResetInvisibleLabels

```
schHiResetInvisibleLabels(
     )
     => t
```

Description

Causes the highlighted labels to either remain visible or reverse to the invisible state in the design. Usable only when editing schematics.

Click each of the highlighted (yellow) labels that you want to make visible after you execute this function.

When you select a label, it remains highlighted but stops blinking, indicating it will remain visible when you quit the command.

When you click again on a label to deselect it, it starts blinking again, indicating it will remain invisible when you quit the command.

Note: To make a label invisible use the Object Property form and change the *Label* drop-down option to *off*.

Arguments

None

Value Returned

Always returns `t`.

schHiReturn

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

Description

Returns up the hierarchy. Usable when editing schematics or symbols after completing a descend or edit-in-place action.

Displays the parent view of the specified cellview.

The parent view is displayed in the current window or an existing window depending on whether you have enabled the *Create New Window When Descending* option on the User Preferences form.

If the current cellview is a multisheet schematic and the index sheet was skipped when descending, the parent cellview of the multisheet schematic will be displayed.

Arguments

w_windowId	Window where the function runs. If not specified, the current window is used.

Value Returned

Always returns `t`.

schHiReturnToTop

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

Description

Returns to the top-level cellview in the hierarchy. Usable when editing schematics or symbols after completing a series of descend commands.

The top cellview is displayed in the current window or an existing window depending on whether you have enabled the *Create New Window When Descending* option on the User Preferences form.

Arguments

w_windowId	Window where the function runs. If not specified, the current window is used.

Value Returned

Always returns `t`.

schHiRotate

```
schHiRotate(
     [ ?nonModal g_nonModal ]
     )
     => t
```

Description

Rotates objects. Usable when editing schematics or symbols.

Operates on the selected set. If the selected set contains multiple objects, you are prompted to click a reference point. If the selected set is empty, you are prompted to point at an object to rotate; that point becomes the reference point. The objects must be fully selected.

If *g_nonModal* is `t` or there are objects in the selected set, the function cancels after the first successful rotation. Otherwise, the function repeats and prompts you to select another object.

Arguments

`?nonModal` *g_nonModal*	Specifies whether the function should be non-modal. Default: `t`

Value Returned

Always returns `t`.

schHiRouteFlightLine

```
schHiRouteFlightLine(
     [ ?routeMethod t_routeMethod ]
     )
     => t
```

Description

Routes logical connections shown as flight lines. Usable only when editing schematics.

If you have not selected the flight line you want to edit, the schematic editor prompts you to select the flight line. The schematic editor can process multiple flight lines. The system routes the selected flight lines.

A *t_routeMethod* of `full` executes an algorithm that creates orthogonal line segments. A *t_routeMethod* of `direct` creates a straight solid line between the two points. If you select multiple wires or flight lines, each wire or flight line is routed in sequence.

Arguments

`?routeMethod` *t_routeMethod*

Method used when routing flight lines; must be enclosed in quotation marks.
Valid Values: `full`, `direct`

Value Returned

Always returns `t`.

Examples

Routes the flight lines using the full routing algorithm.

```
schHiRouteFlightLine( "full" )
```

schHiRouteNamedConnection

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

Description

Starts an interactive command for converting existing wire-by-name connections into wired connections.

Arguments

w_windowId	Window where the function runs. If not specified, the current window is used.

Value Returned

Always returns `t`.

Examples

Start the Route Named Connection command in the current window.

```
schHiRouteNamedConnection()
```

Related Topics

Routing Named Connections

Route Named Connection

Wire-by-Name Connections

schHiSaveCellView

```
schHiSaveCellView(
     )
     => t
```

Description

Saves the design in the current window.

Arguments

None

Value Returned

Always returns `t`.

schHiSelectAll

```
schHiSelectAll(
     )
     => t
```

Description

Opens a form that you use either to add all specified objects to the selected set or to delete all specified objects from the selected set. Usable when editing schematics or symbols.

Arguments

None

Value Returned

Always returns `t`.

schHiSelectByProperty

```
schHiSelectByProperty(
     [ ?select t_select ]
     [ ?propName t_propName ]
     [ ?condOp t_condOp ]
     [ ?propValue t_propValue ]
     )
     => t
```

Description

Adds objects that match specified search criteria in a schematic or symbol view to the selected set or removes objects that match specified search criteria in a schematic or symbol view from the selected set. Usable when editing schematics or symbols. Supports only selection of strings, integers, floating-point numbers, and time. This function does not support other property types.

Searches the schematic or symbol cellview for objects that match the *t_propName*, *t_condOp*, and *t_propValue* arguments.

If no arguments are specified, the Select By Property form appears.

Arguments

?select *t_select*	Specifies if the objects are to be added or removed from the selected set; must be enclosed in quotation marks. Valid Values: select (for add), deselect (for remove)
?propName *t_propName*	The property name used in the search criteria; must be enclosed in quotation marks. If *t_propName* is set to master, *t_propValue* must be *t_libName t_cellName t_viewName* (separated by spaces). Wildcard expressions are not supported if *t_propName* is set to master.
?condOp *t_condOp*	The conditional operator that is applied to the property name and property value for matching; must be enclosed in quotation marks. Valid Values: ==, !=, <=, >=, <, >
?propValue *t_propValue*	The value of the property; must be enclosed in quotation marks. The value may include wildcard expressions.

Value Returned

Always returns `t`.

Examples

Displays the options form with *`t_propName`* set to `instName`, *`t_condOp`* set to `==`, and *`t_propValue`* set to `I1`. All objects that have names set to `I1` are added to the selected set.

```
schHiSelectByProperty( "select" "instName" "==" "I1" )
```

Displays the options form with *`t_propName`* set to `instName`, *`t_condOp`* set to `==`, and *`t_propValue`* set to `I1`. All objects in the selected set that have names set to `I1` are removed from the selected set.

```
schHiSelectByProperty( "deselect" "instName" "==" "I1" )
```

schHiSetSymbolOrigin

```
schHiSetSymbolOrigin(
     )
     => t
```

Description

Relocates the origin point of a symbol and prompts you for a location of the new symbol origin point. All objects in the symbol are moved so when you later place the symbol, it is positioned on the dragging pointer relative to the new origin point.

Usable only when editing symbols.

Arguments

None

Value Returned

Always returns `t`.

schHiShowScope

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

Description

Displays a dialog box describing the current hierarchical scope in the window. Usable when editing or reading schematics or symbols. The scope contains a list of instance names and the corresponding cell names.

Arguments

w_windowId	The window that contains the design whose hierarchical scope you want to display.

Value Returned

Always returns `t`.

schHiSnapToGrid

```
schHiSnapToGrid(
     [ g_windowId ]
     )
     => t / nil
```

Description

Places the objects on the grid in the given window. If the objects are already selected, the *Snap To Grid* command runs in the pre-select mode. If no object is selected in the given window, the command runs in the post-select mode.

Arguments

g_windowId	The window indicating the cellview in which you want to snap the objects to grid. If not specified, the current window is used. This is an optional argument.

Value Returned

`t`	Objects have been snapped to grid successfully.
`nil`	Snapping was unsuccessful.

Examples

The following example can be used for the current window.

```
schHiSnapToGrid()
```

Use this example to specify the specific window where you want to snap the objects to a grid.

```
schHiSnapToGrid( hiGetCurrentWindow() )
```

Related Topics

Snapping to Grid

schHiSolder

```
schHiSolder(
     )
     => t
```

Description

Creates a solder dot over a + or T wire segment. Usable only when editing schematics.

Lets you place a graphical solder dot at an existing T or + connection point. If a wire crosses over another without forming a connection, placing a solder dot at the crossover point forces a connection between the two wires.

Note: When the environment variable `autoDot` is turned on, the *Create – Wire* command will automatically create solder dots at T or + wire connection points.

Arguments

None

Value Returned

Always returns `t`.

schHiSRC

```
schHiSRC(
     [ ?action t_action ]
     )
     => t
```

Description

Sets schematic rules checker (SRC) options, and runs the schematic rules checker. Usable only when editing schematics.

If you set *t_action* to `editOptions`, a form appears on which you can modify the settings of the various schematic rules checks. If online schematic rules checking is inactive, you can also specify `run` and `editOptionsAndRun`.

If you set *t_action* to `run`, the schematic rules checker is run with the current option settings on the current schematic. If you set *t_action* to `editOptionsAndRun`, a form appears on which you can change the rules options.

When all the values have been specified, the schematic rules checker is run on the current schematic. If you do not specify *t_action* or you specify it as `nil`, the function behaves as if you specified `editOptionsAndRun`.

The checks that you run are determined by `ignored`, `warning` (default), and `error` values.

The check is run if the severity is either `warning` or `error`. If you set the severity to `ignored`, that particular schematic rules check is not performed.

All markers generated by this check are indicated with a source of `SRC`. A dialog box displays the total number of errors and warnings detected when the schematic rules checker has completed.

You must correct detected errors before the connectivity in the schematic is validated.

Arguments

`?action` *t_action*	Edits the options, runs the schematic rules checker, or both; must be enclosed in quotation marks. Valid Values: `editOptions`, `run`, `editOptionsAndRun`

Value Returned

Always returns `t`.

Examples

Runs the schematic rules checker when online schematic rules checking is inactive by using the current option settings on the current schematic.

```
schHiSRC( "run" )
```

Displays a form where you edit the options, then runs the schematic rules checker on the current schematic.

```
schHiSRC( "editOptionsAndRun" )
```

schHiStretch

```
schHiStretch(
     [ ?routeMethod t_routeMethod ]
     [ ?useSelSet t_useSelSet ]
     )
     => t
```

Description

Moves objects or partially selected objects and maintains connectivity with rubberband lines. Usable only when editing schematics. You cannot stretch objects between two different cellviews.

The *t_useSelSet* argument controls what is stretched. If this argument is set to `useSelSet` and the selected set contains multiple objects, you are prompted to click a reference point. If the selected set is empty or *t_useSelSet* is `noSelSet`, this function is nonmodal, and you are prompted to click to select the object to stretch and this point becomes the reference point. In either case, the system prompts you to click the destination point.

If you select objects other than instances, paths, schematic pins, or wires, they move instead of stretch.

Drag your objects while clicking on a destination. While you drag your selected object, rubberband lines appear that indicate connectivity between the selected object and the object it was connected to.

After you click the destination, the schematic editor computes new connectivity. The schematic editor routes rubberband lines between the selected objects and their old connection by the specified route method.

Arguments

`?routeMethod` *t_routeMethod*	Method for rerouting stretched wires; must be enclosed in quotation marks. Valid Values: `full`, `direct`, `flight`, `simple`
`?useSelSet` *t_useSelSet*	Specifies whether to stretch the selected set or the object you choose with the mouse; must be enclosed in quotation marks. Valid Values: `useSelSet`, `noSelSet` Default: `useSelSet`

Value Returned

Always returns `t`.

Examples

Stretches selected objects to a destination point leaving flight lines. Flight lines left in a schematic indicate intended connectivity.

```
schHiStretch( "flight" )
```

Stretches selected objects to a destination point using a routing algorithm that generates orthogonal segments to complete the connection path.

```
schHiStretch( "full" )
```

Prompts you to select the object to stretch, even if there are objects in the selected set. Connectivity between the stretched object and other objects is routed using the `full` routing method.

```
schHiStretch( "full" "noSelSet" )
```

schHiSymStretch

```
schHiSymStretch(
     )
     => t
```

Description

Moves partially selected objects in the symbol editor. Can only be used when editing a symbol. Stretching between two different cellviews is not supported. When stretching an edge of a device border, the pin stubs attached to it are not dragged.

If the selected set contains multiple objects, you are prompted to click a reference point. If the selected set is empty, you are prompted to point at an object to stretch; the point becomes the reference point. In either case, the objects are dragged and you are prompted to specify a destination point.

Arguments

None

Value Returned

Always returns `t`.

schHiTree

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

Description

Used to display the design hierarchy listed in text format in the Tree Output window.

You can choose to display the hierarchy from either *Top to bottom*, *Current to bottom*, or *Top to Current*.

Arguments

w_windowId	Window where the function runs. If not specified, the current window is used.

Value Returned

t	Command successfully run.
nil	Command failed to run (perhaps due to current view not being a schematic view).

Examples

```
schHiTree()
```

Related Topics

Displaying Cellview Listings Using the Print Tree

schHiUpdatePinOrder

```
schHiUpdatePinOrder(
     [ g_updateInstLastChanged ]
     )
     => t
```

Description

Creates or modifies the property on the cellview that specifies the ordering of the pins in the current cellview. Usable when editing schematics or symbols.

Important

This function is now replaced by `schHiEditPinOrder`. Use `schHiEditPinOrder` instead of `schHiUpdatePinOrder`.

Operates on the current cellview. It creates or modifies a property on the cellview that describes each pin in the cellview. The importance of the property is to maintain a specific ordering of the pins that can be synchronized against the port ordering of an HDL instance.

Arguments

g_updateInstLastChanged

Specifies whether the time stamp on the cellview is modified.

Value Returned

Always returns `t`.

Related Topics

schHiEditPinOrder

schHiVHDLProperty

```
schHiVHDLProperty(
     )
     => t
```

Description

Edits properties specific to the schematic and VHDL netlister. Usable when editing schematics or symbols.

Operates on the current cellview. It creates or modifies properties for generics, attributes, default scalar and vector data types, library use clauses, and user comments to be used by the schematic editor and VHDL netlister. The netlister uses these properties when generating VHDL text from the cellview.

Arguments

None

Value Returned

Always returns `t`.

schHiUpdatePinsFromView

```
schHiUpdatePinsFromView(
     [ ?viewName t_viewName ]
     )
     => t
```

Description

Updates the pin information in the current cellview from another view of the same cell. This works from schematic to symbol cellview, and vice-versa.

Entering `schHiUpdatePinsFromView()` will display the Update Pins form which allows you to manually select the view you want to update from.

If a pin direction is changed, using the `schHiUpdatePinsFromView()`command, the pin will be re-mastered as appropriate to the new direction, potentially changing the pin shape and label position in the process. This would be equivalent to performing the same task using Edit Object Properties and having to manually make pin direction changes.

Arguments

`?viewName` *`t_viewName`*	Specifies the view that you want to update the current view from, for example “schematic”. This is an optional argument. If a view is not specified, the Update Pins form is displayed, prompting you to select a view.

Value Returned

Always returns `t`.

Examples

Specifies that you want to update the current cellview, for example symbol, with the latest pin information from the cell’s schematic view.

```
schHiUpdatePinsFromView("schematic")
```

Specifies, using keyword arguments, that you want to update the current cellview with the latest pin information from the cell’s symbol view.

```
schHiUpdatePinsFromView(?viewName "symbol")
```

Related Topics

Update Pins

schHiVIC

```
schHiVIC(
     [ ?viewList t_viewList ]
     )
     => t
```

Description

Runs the cross-view checker (VIC) to check the consistency of the interface of one or more cellviews against the cellview you are editing. Usable when editing schematics or symbols.

This function compares the member terminals of the cellview against the member terminals of the views named in *t_viewList*. The check flags export signals that differ between the currently edited view and the signals exported in other views of the same cell. The current cellview and the views you check it against must be compatible with the Cadence® database.

If you do not specify *t_viewList*, a form appears on which you enter the list of views to check.

The types of errors that are reported are for signals exported in one view but not the other and signals whose terminals have different directions in the two views. Use this function to check the consistency between a schematic and the corresponding symbol.

A dialog box displays the total number of errors and warnings detected when the cross-view checker has completed.

Arguments

?viewList *t_viewList*	List of view names to check; must be enclosed in quotation marks. To list several view names, use a space between names as a delimiter. *t_viewList* specifies the names of the cellviews that you want to compare with the currently edited cellview.

Value Returned

Always returns `t`.

Examples

Runs the cross-view checker between the currently edited cellview and the views named `symbol` and `verilog`.

```
schHiVIC( "symbol" "verilog" )
```

schHiVICAndSave

```
schHiVICAndSave(
     )
     => t
```

Description

Runs the cross-view checker (VIC) on the cellview in the current active window and saves the cellview.

Arguments

None

Value Returned

Always returns `t`.

schHiViewToView

```
schHiViewToView(
     [ ?libName t_libName ]
     [ ?cellName t_cellName ]
     [ ?viewNameFrom t_viewNameFrom ]
     [ ?viewNameTo t_viewNameTo ]
     [ ?dataType t_dataType ]
     )
     => t
```

Description

Generates one type of cellview from another.

If you do not specify any argument or specify an argument as `nil`, the Cellview From Cellview form appears so that you can specify the field values.

If the `schViewToPinListReg` list or the `schPinListToViewReg` list does not contain the desired view type and conversion function, you must modify the lists. The lists are defined in your `schConfig.il` file (*your_install_dir*`/samples/local/schConfig.il`). Refer to the following section in your `schConfig.il` file for more information about modifying the `schViewToPinListReg` and `schPinListToViewReg` lists.

Arguments

`?libName` *t_libName*
: Library name that contains the two cellviews; must be enclosed in quotation marks.

`?cellName` *t_cellName*
: Cell name to be used for the two cellviews; must be enclosed in quotation marks.

`?viewNameFrom` *t_viewNameFrom*
: Source view name; must be enclosed in quotation marks.
Valid Values: `schematic`, `symbol`, `functional`, `behavioral`, `system`

`?viewNameTo` *t_viewNameTo*
: Destination view name; must be enclosed in quotation marks.
Valid Values: `schematic`, `symbol`, `functional`, `behavioral`, `system`

`?dataType` *`t_dataType`*	String corresponding to an entry in the `schPinListToViewReg` list that specifies how the created cellview will be generated; must be enclosed in quotation marks.

Value Returned

Always returns `t`.

Examples

Translates the schematic cellview for `xyz` to a symbol cellview.

```
schHiViewToView( "myLib" "xyz" "schematic" "symbol" "Schematic-Symbol" )
```

schHiZoomToSelSet

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

Description

Zooms to selected objects in the current window. If no objects are selected prior to actioning the function, you will be prompted to draw a selection area.

This function is only usable when working with schematics or symbols in the Virtuoso Schematic Editing window.

Arguments

None

Value Returned

`t`	Returns `t` when successful.
`nil`	Returns `nil` when selection not successful.

schHiSetOrigin

```
schHiSetOrigin(
     )
     => t
```

Description

Enables to change the origin point of a symbol or schematic. If you place the cell after specifying the origin point, it is automatically placed at the new origin point.

Arguments

None

Value Returned

Always returns `t`.

schSelectByLine

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

Description

Selects objects by drawing a path across the Virtuoso Schematic Editor window. The objects are highlighted based on what is specified in the Schematic Selection Filter form.

Arguments

None

Value Returned

`t`	Returns `t` when successful.
`nil`	Returns `nil` when the option is used on a window outside the Virtuoso Schematic Editor.

Related Topics

Schematic Selection Filter

schSetSelectOptions

```
schSetSelectOptions(
     )
     => t
```

Description

Sets the filter for the selection function. You can select object types by turning toggle switches on or off. You can also choose partial or full selection. Usable when editing schematics or symbols.

The list of object types is different when editing schematic or symbol cellviews. The function displays the window where you change the object filters and set the selection option to partial or full.

Arguments

None

Value Returned

Always returns `t`.

schSingleSelectPt

```
schSingleSelectPt(
     )
     => t
```

Description

Selects the object under the cursor after first deselecting all objects in the selected set. Usable when editing schematics.

This function is equivalent to the Graphics Editor `mouseSingleSelectPt` function. This function also provides other functions with the identity of the most recently selected object, which is required for extended selection.

Arguments

None

Value Returned

Always returns `t`.

2

Virtuoso Schematic Editor Procedural Interface (PI) Functions

The Virtuoso® Schematic Editor SKILL procedural interface (PI) functions implement primitive actions required by SKILL human interface (HI) functions. Therefore, they are considered lower-level calls. To use PI functions, you must

- Supply all the required parameters
- Use the correct syntax

You can often derive the name of the PI function from the HI function by removing the `Hi` portion of the name. For example, the HI function

```
schHiCreateWire
```

has the corresponding PI function

```
schCreateWire
```

Some PI functions are restricted to the schematic or the symbol cellview. Other functions are restricted to multisheet designs or indexes.

PI functions return either a Boolean `t` or an ID if the function completes successfully. If the function fails or cancels, a Boolean `nil` is returned. If the function does not recognize the argument, the function fails, but you do not receive an error message. If there is a type mismatch, you receive a Cadence® SKILL language error. Lower-level functions do not provide error messages.

Related Topics

Virtuoso Schematic Editor Human Interface (HI) Functions

annLoadAnnotationData

```
annLoadAnnotationData(
     w_windowId
     t_annfileList

     )
     => t / nil
```

Description

Loads the saved annotation setup files into the current schematic design. You can specify multiple annotation setup files to load.

Arguments

w_windowId	Window ID in which the annotation setup is to be loaded.
t_annfileList	List of annotation setup files that needs to be loaded. If you do not specify this argument, the files list is derived from `annotationSetupFileList`.

Value Returned

`t`	Setup files successfully loaded.
`nil`	Command failed.

Examples

```
annSetupsList = "/home/user1/resistorSetup.as /home/user1/dcSetup/pmosSetup.as"
schWindow = hiGetCurrentWindow()

when(schWindow
    annLoadAnnotationData(schWindow annSetupsList)
)
```

annSaveAnnotationData

```
annSaveAnnotationData(
     w_windowId
     t_fileName

     )
     => t / nil
```

Description

Saves the current annotation setup in a file. You can specify the complete file path where the setup file needs to be saved.

Arguments

w_windowId	Window ID in which the annotation setup is to be saved.
t_fileName	File name with the complete path where you want to save the current annotation setup.

Value Returned

t	Setup files successfully saved.
nil	Command failed.

Examples

The following example shows how to save the setup information:

```
fileName = "/home/user1/annSetup/modified_tranSetup.as"
schWindow = hiGetCurrentWindow()

when(window
    annSaveAnnotationData(schWindow fileName)
)
```

defcell

```
defcell(
     t_cellName
     t_pinType
     t_pinName
     defsymbol
     t_viewname
     symbolProps
     t_propName
     g_propValue
     defTermProp
     t_defPinName
     defVisibleProp
     labelAttr
     symbolParam
     f_spacing
     f_length
     f_vLength
     f_hLength
     t_connector
     t_origin
     symbolLabels
     t_labelText
     controlParam
     g_boolean
     pinNumSpec
     t_pinName
     x_pinNumber
     pinLocSpec
     t_leftPinName
     t_rightPinName
     t_topPinName
     t_bottomPinName
     pinGraphicSpec
     t_pinGraph
     pinLogicSpec
     t_posPin
     t_negPin
     clockPins
     t_clockPin
     )
     => t / nil
```

Description

Used with TSG template files to call `lmdefcell`.

Note: `defcell` is classified as a control command in that it is used as a control in a tsg template file. When this template is read, `lmDefCell()` is run based on the `defcell` settings.

Arguments

t_cellName	The cell name portion of a symbol cellview into which the generated symbol is stored; must be enclosed in quotation marks.
t_pinType	The type of pin: must be enclosed in quotation marks. Valid Values: `input`, `output`, `io` By default, all input pins are drawn on the left side of the symbol from top to bottom, all output pins are drawn on the right side of the symbol from top to bottom, and all I/O pins are drawn on top of the symbol from left to right. To move a pin to any other side, specify the pin in the `pinLocSpec` construct. If a pin specified in `pinLocSpec` is also specified in the `output` or `io` constructs, an error message is generated.
t_pinName	The name of the pin: must be enclosed in quotation marks.
defsymbol	Specifies symbol properties and parameters, control parameters, labels, and pin information. These values override applicable defaults.
t_viewName	The view name of the cellview to be created. If you specify *t_viewName*, it must be the first parameter in the `defsymbol` construct. Default: `symbol`
symbolProps	Creates properties on the symbol cellview or properties on sets of terminal pins.
t_propName	The name of the property.

g_propValue	The value of the property to set; can be any of the following: *tnb_value* *tnb_value* [({ { *t_stringEnum* | *t_string* } | { *n_beginRange* | nil } { *n_endRange* | nil } })] time(*t_timeVal*) filename(*t_filename*) ilExpr(*t_ilExpr*) nlpExpr(*t_nlpExpr*).
defTermProp	Defines properties on terminals.
t_defPinName	The pin name to define properties for. Valid Values: `all`, `input`, `output`, `inputOutput`, `top`, `bottom`, `left`, `right`
defVisibleProp	Defines a property and the label attributes information for displaying that property.
labelAttr	Defines the attributes for the label.
symbolParam	Specifies symbol parameters. They can be included in any order without replication. They are *not* stored in the symbol master property list.
f_spacing	The minimum spacing between any pair of adjacent pin wires on a side, rounded to the nearest multiple of snap spacing.
f_length	The length of each pin wire (a pin stub), rounded to the nearest multiple of snap spacing Default: twice the value of the `wireSpacing` field
f_vLength	The minimum length of the vertical sides, rounded to the nearest snap spacing. If the value you supply is less than the computed default, the computed default is used.
f_hLength	The minimum width of the horizontal sides, rounded to the nearest snap spacing. If the value you supply is less than the computed default, the computed default is used.
t_connector	The default pin connector graphic (each entry is mapped into a cellview that contains the graphics for the connector); must be enclosed in quotation marks. Valid Values: `square`, `block`, `circle`
t_origin	The default placement of the symbol origin. Valid Values: `topLeftPin`, `bottomLeftPin`, `centerLeft`

symbolLabels	Define symbol labels. If this construct is not specified, a set of default labels is generated on the symbol.
t_labelText	The text of the label; must be enclosed in quotation marks.
controlParam	Contains control parameters to define execution behavior. This control parameter is *not* stored in the property list of the symbol master.
g_boolean	Use only when the specified symbol master already exists in the library. When set to `t`, TSG asks several questions interactively. When set to `nil` (the default), TSG assumes an existing symbol should be overwritten. It does *not* prompt you for permission to overwrite an existing symbol cellview. Valid Values: `t`, `nil` Default: `nil` This control parameter is the only one that can be included in this construct and is *not* stored in the property list of the symbol master.
pinNumSpec	This construct is most often used for printed circuit board design libraries.
t_pinName	A pin name; must also be declared in `input`, `output`, or `io`.
x_pinNumber	A pin number.
pinLocSpec	Allows you to override default pin placement locations; specifies the pin location in any order without replication.
t_leftPinName	The names of pins to draw on the left of the symbol; must be enclosed in quotation marks. Must also be specified in the `input`, `output`, or `io` construct. Other pins can be drawn on the left of the symbol. Pins specified in `input` are also drawn on the left of the symbol. A pin specified in `input` but *not* specified in `pinLocSpec` is drawn on the left before those in `leftPins`. A pin name in `leftPins` cannot be specified again in the other three subconstructs.

t_rightPinName	The names of pins to draw on the right of the symbol; must be enclosed in quotation marks. Must also be specified in the `input`, `output`, or `io` construct. Other pins can be drawn on the right of the symbol. Pins specified in `output` are also drawn on the right of the symbol. A pin specified in `output` but *not* specified in `pinLocSpec` is drawn on the right before those in `rightPins`. A pin name in `rightPins` cannot be specified again in the other three subconstructs.
t_topPinName	The names of pins to draw on the top of the symbol; must be enclosed in quotation marks. Other pins can be drawn on the top of the symbol. Pins specified in `io` are also drawn on the top of the symbol. A pin specified in `io` but not in `pinLocSpec` is drawn on the top before those in `topPins`. A pin name in `topPins` cannot be specified again in the other three subconstructs.
t_bottomPinName	The names of pins to draw on the bottom of the symbol; must be enclosed in quotation marks. A pin name in `bottomPins` cannot be specified again in the other three subconstructs.
pinGraphicSpec	Identifies special pin graphics, such as negation bubbles and clock indicators, to be drawn with the specified pins.
t_pinGraph	Identifies special pin graphics, such as negation bubbles and clock indicators. Valid Values: `actHi`, `actLo`, `ieeeActLo`, `clock`, `actLoClock` Default: `actHi` Each of these values must match an entry in the `schConfig.il` file.
pinLogicSpec	Using this construct is *not recommended*, but it remains supported for backward compatibility. The `pinGraphicSpec` construct is the preferred method for specifying special pin graphics. Specifies negation indicators.
t_posPin	A list of pin names that indicate the positive logic type of the pins; must be enclosed in quotation marks.

t_negPin	A list of pin names that indicate the negative logic pin type; must be enclosed in quotation marks. Negative pins appear in the symbol as a bubble.
clockPins	Using this construct is *not recommended*, but it remains supported for backward compatibility. The `pinGraphicSpec` construct is the preferred method for specifying special pin graphics. Specifies the display of clock indicators.
t_clockPin	A list of pin names that indicate clock pins. Any pins must also be specified in `input`, `output`, or `io`. Clock pins are designated by a small triangle next to the pin.

Value Returned

`t`	Command successfully run.
`nil`	Command failed.

Examples

```
defcell("count4"
    input("clock" "reset")
    output("a0" "a1" "a2" "a3")
    defsymbol(
       symbolProps(
          partName = "count4" ; cellview prop with valueType=string
          pinNum   = (4 (0 24))
          timeVal  = time("Jan 1 12:00:00 1993")
          color    = ("red" ("red" "green" "blue"))
          defTermProp(
              input Iih = 1.24 ) ; all input pins get property "Iih".
          defTermProp(
              all b=6 )       ; all pins get property "b".
       symbolParam(
          wireSpacing  = 0.125000
          wireLength   = 0.250000
          vSideLength  = 0.000000
          hSideLength  = 0.000000
          origin       = topLeftPin
          pinConnector = "block"
       )
       symbolLabels(
          defLabel( name("[@partName]")
              location( "(xleft + xright)/2:(ytop + ybottom)*3/4")
              labelType(NLPLabel)
              layer(device)
              purpose(label)
           )
           defLabel( name("[@instanceName]")
              location( "xleft:(ytop + ybottom)/2")
              labelType(NLPLabel)
              justification(upperLeft)
              layer(instance)
              purpose(label)
              apply(cellview)
              fontHeight(0.1)
           )
           defLabel( name("{pinName}")
              location( "1.15*stubLength:0")
              justification(centerLeft)
              apply(left)
           )
           defLabel( name("[@p_{pinName}]")
              location( "-stubLength/2:0.03125")
              labelType(NLPLabel)
              layer(pin)
              purpose(annotate)
              apply(right)
           )
       )
       pinNumSpec( "a0":16  "a1":17  "a2":18 )
       pinGraphicSpec( "reset":ieeeActLo  "clock":clock )
    )
)
```

Related Topics

tsg

Text-to-Symbol Generator

g_propValue

labelAttr

hsmDeselect

```
hsmDeselect(
     [ ?type d_type ]
     [ ?path g_path ]
     [ ?name g_name ]
     [ ?spec l_spec ]
     )
     => t / nil
```

Description

Deselects named objects in the design hierarchy.

Arguments

?type *d_type*	The object type to be deselected. Objects currently supported are: instances, nets, pins, terminals, paths, vias, PRBoundary, areaBoundary, clusterBoundary, snapBoundary, figGroups and clusters.
?path *g_path*	The hierarchical path can be either a string, window or list.
?name *g_name*	The name of the object to be deselected. Can be either a string for a single object or a list of strings for multiple objects.

?spec *l_spec*	The general specification API.
	For example:
	`when( ss=hsmSelectedSet(?type 'net)`
	`hsmDeselect(?spec ss)`
	`)`

Value Returned

t	Object successfully deselected.
nil	Command failed.

Examples

```
hsmDeselect(?type 'inst ?path hiGetCurrentWindow ?name list("I3" "I4"))
hsmDeselect(?type 'inst ?path "(amsPLL.vco:schematic)/IN23" ?name list("I3" "14"))
```

hsmGetSelectedSet

```
hsmGetSelectedSet(
     [ ?top g_top ]
     [ ?type d_type ]
     [ ?pathStyle g_pathStyle ]
     [ ?includeExtra g_includeExtra ]
     [ ?cellview d_cellview ]
     [ ?path g_path ]
     )
     => list / nil
```

Description

Gets the hierarchical selected set.

Arguments

`?top` *g_top*	The top cellview or window.
`?type` *d_type*	The object type to be retrieved. Objects currently supported are: instances, nets, pins, terminals, paths, vias, PRBoundary, areaBoundary, clusterBoundary, snapBoundary, figGroups and clusters.
`?pathStyle` *g_pathStyle*	The path style. For example: ■ `list` - gives the path as a list of lists ■ `string` - gives a simple path as a simple string e.g. “(L.C:V)/I1/I2/I3”) ■ `qualified` - gives the path as a full path including library, cell and view
`?includeExtra` *g_includeExtra*	Lists extra details on selected objects. This is `nil` by default.

?cellview *d_cellview*	Enter either a db cellview or a string of the form “lib cell view”, so that it only returns the selections that are in that cellview. **Note:** If you have the same cellview open in multiple windows or tabs, at different levels of the hierarchy, you will get multiple entries for all of the difference hierarchies.
?path *g_path*	Enter a hierarchical path that can be either a string, window or list of lists, and it will return only the those selections at the lowest level of the path.

Value Returned

list	The returned hierarchical selected set.
nil	Commands failed.

Examples

Will get everything.

```
hsmGetSelectedSet()
```

Will get just instances in the hierarchy starting at the given cell view.

```
hsmGetSelectedSet(?type 'inst)
```

Will get nets and instances in the hierarchy starting at the given cell view.

```
hsmGetSelectedSet(?type list('inst 'net))
```

Will get extra details on selected objects.

```
hsmGetSelectedSet(?includeExtra t)
```

Related Topics

hsmSelect

hsmSelect

```
hsmSelect(
     [ ?type d_type ]
     [ ?path g_path ]
     [ ?name g_name ]
     [ ?spec l_spec ]
     )
     => t / nil
```

Description

Selects named objects in the design hierarchy.

Arguments

?type *d_type*	The object type to be selected. Objects currently supported are: instances, nets, pins, terminals, paths, vias, PRBoundary, areaBoundary, clusterBoundary, snapBoundary, figGroups and clusters.
?path *g_path*	The hierarchical path can be either a string, window or list.
?name *g_name*	The name of the object to be deselected. Can be either a string for a single object or a list of strings for multiple objects.

?spec *l_spec*	The general specification API.
	For example:
	`when( ss=hsmSelectedSet(?type 'net`
	`hsmDeselect(?spec ss)`
	`)`

Value Returned

t	Object successfully selected.
nil	Command failed.

Examples

```
hsmSelect(?type 'inst ?path hiGetCurrentWindow ?name list("I3" "I4"))

hsmSelect(?type 'inst ?path "(amsPLL.vco:schematic)/IN23" ?name list("I3" "14"))
```

opcAddListToSet

```
opcAddListToSet(
     o_set
     l_objects
     )
     => t / nil
```

Description

Adds a list of objects to the specified set.

Arguments

o_set	Name of the operating collections (OPC) set that is updated by adding the objects.
l_objects	List of objects to be added.

Value Returned

t	All objects added to the set successfully.
nil	Failed to add the objects.

Examples

To add the instances to a set, use the following command:

```
cv = geGetEditCellView()
instances = cv~>instances
set = opcFindSet(cv "mySet")
opcAddListToSet(set instances)
;; check how many objects in the set
length(set~>objects)
```

opcAddObjectToSet

```
opcAddObjectToSet(
     o_set
     d_object
     )
     => t / nil
```

Description

Adds an object to the specified set.

Arguments

o_set	Name of the OPC set that is updated by adding the objects.
d_object	Object to be added.

Value Returned

t	Object added to the set successfully.
nil	Failed to add the object.

Examples

To add an instance to a set, use the following command:

```
cv = geGetEditCellView()
inst = car(cv~>instances)
set = opcFindSet(cv "mySet")
opcAddObjectToSet(set inst)
;; check for the object in the set
set~>objects
```

opcAllSetsInCellView

```
opcAllSetsInCellView(
     d_cellview
     )
     => l_sets
```

Description

Returns a list of all the sets present in the specified cellview.

Arguments

d_cellview	Name of the cellview.

Value Returned

l_sets	List of sets in the given cellview.

Examples

To list all the sets in `cv`, use the following commands:

```
cv = geGetEditCellView()
sets = opcAllSetsInCellView(cv)
when(length(sets) != 0
   info("Cellview %s.%s.%s contains.\n" cv~>libName cv~>cellName cv~>viewName)
   foreach(s sets
      info("Set %s with %d objects.\n" s~>name length(s~>objects))
)
)
```

opcClearSet

```
opcClearSet(
     o_set
     )
     => t / nil
```

Description

Removes all objects from a set without modifying the specific objects.

Arguments

o_set	Name of the OPC set from which the objects are being removed.

Value Returned

t	All objects removed from the set successfully.
nil	Failed to remove all objects.

Examples

To remove mySet from the current cellview current, set the following commands:

```
cv = geGetEditCellView()
set = opcFindSet(cv "mySet")
;; confirm that there are objects in the set
length(set~>objects)
opcClearSet(set)
;; confirm that all objects have been removed
length(set~>objects)
```

opcCreateCondition

```
opcCreateCondition(
     ?attrName s_attrName
     ?opName g_opName
     ?value t_value
     )
     => nil / list
```

Description

Defines a condition to be used in Navigator custom queries.

Arguments

`?attrName` *`s_attrName`*	Attribute for the condition, for example, `Name`, `Bits`, `Placement`, `Ignored`, `Global`, `Signal`.
`?opName` *`g_opName`*	Operator for the condition with a specified attribute. It can be an arithmetical operator, for example, `=`, `!=`, `<`, `<=`, `>`, `>=`. It can also be a string predicate, for example, `Equals` or `Contains`.
`?value` *`t_value`*	Value to be applied on the attribute with the specified operator, for example, `clock`, `true`, `8`.

Value Returned

A SKILL list containing the attribute, operator, and value as elements.

Examples

Define a condition for names that contain the word `clock`.

```
condition = opcCreateCondition(?attrName 'Name ?opName "Contains" ?value "clock")
```

The returned value is (`"Name" "Contains" "clock"`) and can be used in defining expressions.

Related Topics

opcCreateExpression

opcCreateExpression

```
opcCreateExpression(
     ?dataSource s_dataSource
     ?dataType t_dataType
     [ ?conditions l_conditions ]
     )
     => nil / list
```

Description

Defines an expression to be used in Navigator custom queries.

Arguments

`?dataSource` *`s_dataSource`*

Design object to be used in the expression, for example, `Instances`, `Nets`, `Pins`.

`?dataType` *`t_dataType`*

Design object type. Only required for user-specified data. Example: `instance`, `net`, `pin`.

`?conditions` *`l_conditions`*

List of conditions that must be met for the objects in the data source to be accepted as valid by the expression. The conditions are combined using logical AND. Conditions are created using `opcCreateCondition`.

Value Returned

A SKILL list containing the data source and optionally the data type and list of conditions as elements.

Examples

Define an expression that can be used to retrieve iterated instances that are of size 2 and their name contains `X`.

```
expression = opcCreateExpression(?dataSource 'Instances ?conditions
list(opcCreateCondition(?attrName 'Bits ?opName "=" ?value "2")
opcCreateCondition(?attrName 'Name ?opName "Contains" ?value "X")))
```

The returned value is `("Instances" "instance" (("Bits" "=" "2") ("Name" "Contains" "X")))` and can be used in defining queries.

Related Topics

opcClearSet

opcCreateQuery

opcCreatePersistentSet

```
opcCreatePersistentSet(
     d_cellview
     t_name
     )
     => o_set / nil
```

Description

Creates an OPC set that has a persistent storage in the cellview.

Arguments

d_cellview	Name of the cellview that stores the selected members of the set as well as the sets.
t_name	Name of the OPC set.

Value Returned

o_set	The newly created OPC set.
nil	Failed to create the set.

Examples

Suppose `mySet` is required even after the cellview has been closed. Use the following commands assuming that the cellview is opened for edits.

```
cv = geGetEditCellView()
set = opcCreatePersistentSet(cv "mySet")
if(set then
   info("Successfully created set %s.\n" set~>name)
else
   info("Failed to create set.\n")
)
```

opcCreateQuery

```
opcCreateQuery(
     ?name t_name
     [ ?category t_category ]
     [ ?appName t_appName ]
     [ ?expressions l_expressions ]
     )
     => t / nil
```

Description

Defines a query that can be instantiated in a cellview to generate a set of objects matching the expressions that constitute the Navigator custom query.

Arguments

t_name	Name of the query. The name is displayed where queries can be instantiated, for example, in the Navigator assistant. Example: `"MultiBit"`
t_category	Name of the category the query will be placed under. If not specified `"Miscellaneous"` is used. Example: `"Design Exploration"`
t_appName	Name of the application where the query can be applied. If you do not specify this, you can apply the query in any application. Example: `schematic`
l_expressions	List of expressions that define the content in terms of design objects for the query. Expressions should be created using `opcCreateCondition`.

Value Returned

`t`	Query is created successfully.
`nil`	Failed to create query.

Examples

Create a query containing iterated instances and buses.

```
condition = opcCreateCondition(?attrName 'Bits ?opName "!=" ?value "1")

instExpression = opcCreateExpression(?dataSource "Instances" ?conditions
list(condition))

netExpression = opcCreateExpression(?dataSource "Nets" ?conditions
list(condition))


opcCreateQuery(?name "MultiBit" ?category "Design Exploration" ?expressions
list(instExpression netExpression))
```

Related Topics

opcCreateCondition

opcCreateTransientSet

```
opcCreateTransientSet(
     d_cellview
     t_name
     )
     => o_set / nil
```

Description

Creates an OPC set that is deleted when the cellview is removed from memory. Such sets can be created even when the cellview is read-only.

Arguments

d_cellview	Name of the cellview from where the members of the set should be selected.
t_name	Name of the OPC set.

Value Returned

o_set	The newly created OPC set.
nil	Failed to create the set.

Examples

Suppose a set called `mySetTransient` is required to store objects but only as long as the cellview is opened. To achieve this, you can use the following command:

```
cv = geGetEditCellView()
set = opcCreateTransientSet(cv "mySetTransient")
if(set then
   info("Successfully created set %s.\n" set~>name)
else
   info("Failed to create set.\n")
)
```

opcDeleteQuery

```
opcDeleteQuery(
     t_name
     )
     => t / nil
```

Description

Deletes the definition of a query along with any sets that have been created using the query.

Arguments

t_name	Name of the query to be deleted.

Value Returned

t	Query is deleted successfully.
nil	Failed to delete query.

Examples

Use the following command to delete the query called MultiBit:

```
opcDeleteQuery("MultiBit")
```

opcDestroySet

```
opcDestroySet(
     o_set
     )
     => t / nil
```

Description

Removes the set from memory and storage.

Arguments

o_set	OPC set to be removed.

Value Returned

t	The OPC set destroyed successfully.
nil	Failed to destroy the set.

Examples

Use the following commands to destroy `mySet`:

```
cv = geGetEditCellView()
set = opcFindSet(cv "mySet")
unless(opcDestroySet(set)
  info("Set %s cannot be destroyed.\n" set~>name)
```

opcEditQuery

```
opcEditQuery(
     t_name
     )
     => t / nil
```

Description

Edits the definition of a query in a modal popup form.

Arguments

t_name	Name of the query to be edited.

Value Returned

t	Query is edited successfully.
nil	Failed to edit query or query cannot be found.

Examples

Use the following command to edit the query called MultiBit:

```
opcEditQuery("MultiBit")
```

opcFindSet

```
opcFindSet(
     d_cellview
     t_name
     )
     => o_opcSet / nil
```

Description

Searches for the specified OPC set name in the given cellview.

Arguments

d_cellview	Name of the cellview to be searched.
t_name	Name of the OPC set to be searched for.

Value Returned

o_opcSet	The name of the OPC set.
nil	Failed to find the specified set.

Examples

To find `mySet` from the cellview currently selected, set the following commands:

```
cv = geGetEditCellView()
name = "mySet"
set = opcFindSet(cv name)
when(set
    info("Set %s contains %d objects\n." set~>name length(set~>objects))
```

opcInstantiateQuery

```
opcInstantiateQuery(
     ?query t_query
     ?cv d_cv
     [ ?category t_category ]
     )
     => t / nil
```

Description

Instantiates a query in a cellview. If a category is specified, the query is instantiated in the specified category. Otherwise, the query is instantiated in the same category as the query.

Arguments

`?query` *t_query*	Name of the query to instantiate.
`?cv` *d_cv*	Name of the cellview where the query is instantiated.
`?category` *t_category*	Name of the category in which to instantiate the query.

Value Returned

`t`	Query is instantiated successfully.
`nil`	Failed to instantiate query.

Examples

Instantiate a query.

```
opcInstantiateQuery(?query "aQuery" ?cv gec() ?category "aCategory")
opcInstantiateQuery(?query "anotherQuery" ?cv gec())
```

opcLoadQueries

```
opcLoadQueries(
     [ ?file t_file ]
     )
     => t / nil
```

Description

Loads all the queries defined in the specified file.

Arguments

`?file t_file`	The file from which the queries are loaded. If no file is provided, a form is displayed where you can specify the file.

Value Returned

`t`	Queries loaded successfully.
`nil`	Failed to load the queries because an invalid file was specified.

Examples

Load queries from the file named `aFile.il`.

```
opcLoadQueries(?file "aFile.il")
```

Load queries without the filename specified. The form is displayed.

```
opcLoadQueries()
```

Related Topics

opcSaveQueries

opcRefreshSet

```
opcRefreshSet(
     d_setId
     )
     => t / nil
```

Description

Refreshes the content of an operating collections set derived from a query containing SKILL customization.

Arguments

d_setId	ID of the set to be refreshed.

Value Returned

t	Set refreshed successfully.
nil	Failed to refresh the set.

Examples

Consider a cellview in which there is a set defined that includes instances with a specific value for a CDF parameter. Assume that the set is called `InstancesHalfCap`. Use the following commands to refresh the set in the cellview `cv` after editing the CDF:

```
hCap = opcFindset(cv "InstancesHalfCap")

opcRefreshSet(hCap)
```

opcReleaseSet

```
opcReleaseSet(
     d_cellview
     )
     => t / nil
```

Description

Removes from cache the sets contained in the specified cellview, but retains them on the hard disk. The function is intended to release memory when critical operations need to be run.

Arguments

d_cellview	Name of the cellview containing the sets to be released.

Value Returned

t	Sets were released successfully.
nil	Failed to release the sets.

Examples

```
cv = geGetEditCellView()
opcReleaseSet(cv)
```

opcRemoveObjectFromSet

```
opcRemoveObjectFromSet(
     o_set
     d_object
     )
     => t / nil
```

Description

Removes an object from the specified set.

Arguments

o_set	The OPC set that is being updated by removing the object.
d_object	Object to be removed.

Value Returned

t	Object removed from the set successfully.
nil	Failed to remove the object.

Examples

To remove an object from the given set, run the following commands:

```
cv = geGetEditCellView()
inst = car(cv~>instances)
set = opcFindSet(cv "mySet")
opcAddObjectToSet(set inst)
;; confirm that object is in the set
set~>objects
opcRemoveObjectFromSet(set inst)
;; confirm that object has been removed from the set
set~>objects
```

opcSaveQueries

```
opcSaveQueries(
     [ ?file t_file ]
     ?queries l_queries
     )
     => t / nil
```

Description

Saves the list of queries in the specified file.

Arguments

`?file` *`t_file`*	The file listing the queries to be saved. If no file is provided, a form is displayed where you can specify the file.
`?queries` *`l_queries`*	List of query names to be saved.

Value Returned

`t`	Queries saved successfully.
`nil`	Failed to save the queries because the list was empty or the file was invalid.

Examples

Save queries named `Custom1` and `Custom2` in the file named `aFile.il`.

```
opcSaveQueries(?file "aFile.il" ?queries list("Custom1" "Custom2"))
```

Save query named `aQuery`. Because no filename is specified, the form is displayed.

```
opcSaveQueries(?queries list("aQuery"))
```

Related Topics

opcLoadQueries

schAddIgnoreProp

```
schAddIgnoreProp(
     t_name
     t_prompt
     t_type
     t_value
     g_enable
     )
     => t / nil
```

Description

Registers an ignore property. You can view the set of registered ignore properties on the *Ignore Properties* tab by clicking *Options – Editor* in the schematic window.

Arguments

t_name	Name of the property to be registered.
t_prompt	Prompt name (if any) of the ignore property that is to be added.
t_type	Type of the property is string or Boolean.
t_value	Value of the ignore property.
g_enable	Enable or disable the property using `t` or `nil`.

Value Returned

`t`	Ignore property was registered successfully.
`nil`	Ignore property was not registered.

Examples

The `nlAction` property is registered.

```
schAddIgnoreProp( ?name "nlAction" ?prompt "ignore all simulators" ?type "string"
?value "ignore" ?enable t )
```

schAlign

```
schAlign(
     l_list
     s_justify
     )
     => t / nil
```

Description

Aligns the objects in the specified direction in the schematic or symbol view.

Arguments

l_list	List of database objects to be aligned.
s_justify	Direction of alignment, such as left, right, horizontal, vertical, top, or bottom.

Value Returned

t	Alignment is done successfully.
nil	The alignment was unsuccessful.

Examples

```
schAlign( geGetSelSet() 'left )
schAlign( geGetSelSet() 'right )
```

Related Topics

Aligning

schAttachLibToPackageTech

```
schAttachLibToPackageTech(
     t_libName
     )
     => d_DBId / nil
```

Description

Attaches a package technology to the given library.

Arguments

t_libName	Name of the library which is to be attached to the package technology.

Value Returned

d_DBId	Returns the database ID of the technology file.
nil	The technology library or file does not exist.

Examples

Attaches the given window's library to the package fabric technology and displays the corresponding message on the CIW.

```
schAttachLibToPackageTech
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
   libName = cellView->libName
   if( tech = schAttachLibToPackageTech(libName) then
     print("Library successfully attached to Package Technology.")
   else
     print("Unable to attach Library to Package Technology.")
   )
  )
)
```

schCheck

```
schCheck(
     d_cvId
     )
     => l_errors
```

Description

Performs a check on the specified cellview. This includes extracting connectivity, running the schematic rules checker, and running the cross-view checker. You must have write permission to any cellview that is to be checked. The given cellview ID can be read-only or editable schematic.

This function uses the following environment settings:

- `updateConn` specifies whether connectivity extraction is performed
- `runSRC` specifies whether the schematic rules checker is run
- `runVIC` specifies whether the cross-view checker is run

The schematic rules checker uses a large set of environment settings that control the checks run. For a list of these settings, see schSRC.

Note: You can run the function schClearConn to remove existing schematic connectivity on the cellview before restarting the extraction using `schCheck`.

Arguments

d_cvId	Cellview ID of the schematic to check.

Value Returned

l_errors	A list containing the total number of errors and warnings encountered. This includes errors and warnings from both the schematic rules checker and the cross-view checker.

Examples

```
cvId = dbOpenCellViewByType( "mylib" "top" "schematic" "" 'a )
errs = schCheck( cvId )
nErrors = car( errs )
nWarns = cadr( errs )
```

Related Topics

schSRC

schClearConn

schCheckHier

```
schCheckHier(
     d_cvId
     t_viewNames
     t_refLibs
     [ l_instViewListTable ]
     )
     => l_errors / nil
```

Description

Performs a check of the hierarchy that starts at the given cellview.

The hierarchy traversed is defined by *t_viewNames*. Usually, the hierarchy is confined to the library of the given cellview, but you can specify a list of reference libraries to process if the hierarchy extends beyond the current library. You must have write permission to any cellview that is to be checked.

This function uses the following environment settings:

- `checkAlways` specifies whether to check every cellview regardless of the extraction status

  When `nil`, cellviews are checked only if they need it.

- `updateConn` specifies whether connectivity extraction is performed on all schematics encountered

- `runSRC` specifies whether the schematic rules checker is run on all schematics encountered

- `runVIC` specifies whether the cross-view checker is run on all cellviews encountered

- `checkHierSave` specifies whether processed cellviews are automatically saved

  If `nil`, you must explicitly save and close the cellviews processed, or any updates are lost.

- `saveAction` specifies what to do for those cellviews containing errors when `checkHierSave` is `t`

  Valid values are `Save`, `No Save`, and `Ask Me`.

Arguments

d_cvId	Cellview ID of the root schematic from which to begin the hierarchical check. The given cellview ID can be read-only or editable schematic. If a cellview contains any of the following property values, it is not processed:
	`nlAction == { ignore | stop }`
	`schType == { border | patchCord | ripper | noSchEdit }`

t_viewNames	String containing the list of view names to use to control the hierarchy traversal; must be enclosed in quotation marks.
t_refLibs	String containing the list of reference libraries to process in addition to the library of the given cellview; must be enclosed in quotation marks.
l_instViewListTable	List specifying the instance view list table to use if instance-based switching is required. This list contains sublists that map a logical name to a view name list. If an instance is encountered that has an `instViewList` property whose value matches one of the logical names in the instance view list table, the view name list associated with the logical name is used for the hierarchical switch for that instance.

Value Returned

l_errors	A list that contains sublists of the ID of the cellview and the number of errors encountered.
`nil`	No errors are found in the hierarchy.

Examples

Checks the hierarchy starting at `top schematic` in the library `mylib` where the traversal is controlled by the given view name list.

```
cvId = dbOpenCellViewByType( "mylib" "top" "schematic" "" 'a )
schSetEnv( "checkHierSave" t )
schSetEnv( "saveAction" "Save" )
errs = schCheckHier( cvId "schematic cmos_sch" "" )
```

If errors are encountered, `errs` is a list of cellview, number of errors pairs. You can process this list as follows:

```
foreach( x errs
    info( "%s %s %s has %d error(s).\n" car(x)~>lib~>name
        car(x)~>cellName car(x)~>viewName cadr(x))
)
```

schCheckHierConfig

```
schCheckHierConfig(
     h_cfgId
     [ v_pathVector [ g_refLibs ] ]
     )
     => l_errors / nil
```

Description

Performs a check of the hierarchy. The check starts with the top cellview that is specified in the given hierarchy configuration. The hierarchy traversed is defined by information in this hierarchy configuration. Usually, the hierarchy is confined to the library of the given cellview, but you can specify a list of reference libraries to process if the hierarchy extends beyond the current library.

You must have write permission to any cellview that is to be checked.

This function uses the following environment settings:

- `checkAlways` specifies whether to check every cellview regardless of the extraction status

  When `nil`, cellviews are checked only if they need it.

- `updateConn` specifies whether connectivity extraction is performed on all schematics encountered
- `runSRC` specifies whether the schematic rules checker is run on all schematics encountered
- `runVIC` specifies whether the cross-view checker is run on all cellviews encountered
- `checkHierSave` specifies whether processed cellviews are automatically saved

  If `nil`, you must explicitly save and close the cellviews processed, or any updates are lost.

- `saveAction` specifies what to do for those cellviews containing errors when `checkHierSave` is `t`

  Valid values are `Save`, `No Save`, and `Ask Me`.

The given cellview ID must be an editable schematic. If a cellview contains any of the following property values, it is not processed:

```
nlAction == { ignore | stop }
schType == { border | patchCord | ripper | noSchEdit }
```

Arguments

h_cfgId	The ID of a hierarchy configuration that specifies an expansion.
v_pathVector	The ID of a hierarchy configuration path vector. If not specified, the traversal starts from the top cellview specified in the configuration. Otherwise, the traversal starts from the current cellview defined by this argument.
g_refLibs	A list of library names, or a string containing a list of space-separated library names.

Value Returned

l_errors	A list of the errors encountered during the checking of the hierarchy as defined by the given configuration object. Each element in the list contains the cellview ID and the number of errors encountered; no information is generated if only warnings were encountered.
`nil`	No errors are found in the hierarchy.

Examples

```
cfgId = deGetConfigId( getCurrentWindow( ))
schSetEnv( "checkHierSave" t )
schSetEnv( "saveAction" "Save" )
errs = schCheckHierConfig( cfgId )
cfgId = deGetConfigId( getCurrentWindow( ))
path = deGetVector( getCurrentWindow( ))
errs = schCheckHierConfig( cfgId path "libA libB" )
```

If errors are encountered, `errs` is a list of cellview/number-of-errors pairs. You can process this list as follows:

```
foreach( x errs
    info( "%s %s %s has %d error(s).\n" car(x)~>lib~>name
    car(x)~>cellName car(x)~>viewName cadr(x))
)
```

schClearConn

```
schClearConn(
     d_cvId
     )
     => t / nil
```

Description

Removes the schematic connectivity from a specified cellview to restart extraction using schExtractConn.

This function does the following:

- Deletes all markers.
- Deletes all non-terminal nets.
- Detaches all shapes from the remaining terminal nets.
- Breaks all explicit net equivalence on nets.
- Removes any inherited net expressions attached on nets.
- Detaches instance pins from terminal nets.

Arguments

d_cvId	ID of the cellview that the schematic connectivity is to be cleared.

Value Returned

t	All schematic connectivity was successfully removed from the cellview.
nil	Failed to clear the schematic connectivity as the cellview is not a valid schematic or symbol cellview.

Examples

Removes the schematic connectivity from the current cellview.

```
cv = geGetEditCellView()
```

```
schClearConn(cv)
```

Related Topics

schExtractConn

schCloneSymbol

```
schCloneSymbol(
     d_cvId
     d_masterId
     l_origin
     t_orient
     )
     => t / nil
```

Description

Copies or clones graphics from an existing symbol into the target symbol cellview with the given location and orientation.

Arguments

d_cvId	Cellview ID of the editable symbol cellview in which to place copied graphics.
d_masterId	ID of the clone master cellview, which can be accessed using several different methods, such as an explicit call to `dbOpenCellViewByType`.
l_origin	Location to place the clone.The origin of the instance master is used as the reference point.
t_orient	Orientation to give the clone placement; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, `MYR90`

Value Returned

`t`	Graphics were copied or cloned from an existing symbol into the target symbol cellview with the given location and orientation.
`nil`	Copying or cloning was unsuccessful.

Examples

Clones an `inverter` symbol from the sample library in the specified symbol cellview. The cloned graphics are placed at 0,0 with an `R0` orientation.

```
symbolId = dbOpenCellViewByType( "sample" "inv" "symbol" "" 'r )
schCloneSymbol( cvId symbolId 0:0 "R0" )
```

schCmdOption

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

Description

Cycles through a predefined set of values. By default, this function is bound to the middle mouse button. When you click the right mouse button during an active command, the command applies the next value in the predefined set.

You can customize the predefined set of values by making calls to `schSetCmdOption`.

Arguments

None

Value Returned

`t`	Cycled through a predefined set of values.
`nil`	Cycling through values was unsuccessful.

schComputePinRef

```
schComputePinRef(
     d_cellView
     [ ?reportFile t_reportFile ]
     [ ?display t_display ]
     [ ?formatString t_formatString ]
     [ ?reportDups t_reportDups ]
     [ ?sortByDir t_sortByDir ]
     [ ?separator t_separator ]
     [ ?inputDesignator t_inputDesignator ]
     [ ?outputDesignator t_outputDesignator ]
     [ ?ioDesignator t_ioDesignator ]
     [ ?charsPerLine x_charsPerLine ]
     )
     => t / nil
```

Description

Creates offsheet pin references for multisheet designs. The pin references can be displayed in the schematic next to each pin or in a report file. This function creates an offsheet pin reference report that lists each pin followed by a list of all other locations of this pin. The pin references can also be displayed in the schematic next to each pin.

Arguments

d_cellView	Cellview of the index schematic or any sheet in a multisheet design to be cross-referenced; must be enclosed in quotation marks.
?reportFile *t_reportFile*	File in which to output the cross-reference report; specify `nil` for no report. Default: `" "`
?display *t_display*	Set to `on` to display cross-references in schematic, set to `off` to remove cross-references in schematics if they exist; must be enclosed in quotation marks. Default: `on`
?formatString *t_formatString*	

Controls the cross-reference format. You can build the cross-reference format using any combination of the following in any order:

sheetNumber zone referenceName direction

Default: `schGetEnv("pinRefFormat")`

`?reportDups` *t_reportDups*

Set to `off` to suppress reporting of duplicate pin references found within the same zone; must be enclosed in quotation marks.
Default: `off`

`?sortByDir` *t_sortByDir*

Sets whether pin reference sorting is by direction or sheet number; must be enclosed in quotation marks. Set this argument to `on` to sort by direction.
Default: `off`

`?separator` *t_separator*

String used to separate pin references; must be enclosed in quotation marks.
Default: `,`

`?inputDesignator` *t_inputDesignator*

String used to designate input pins; must be enclosed in quotation marks.
Default: `i`

`?outputDesignator` *t_outputDesignator*

String used to designate output pins; must be enclosed in quotation marks.
Default: `o`

`?ioDesignator` *t_ioDesignator*

String used to designate IO pins; must be enclosed in quotation marks.
Default: `io`

`?charsPerLine` *x_charsPerLine*

Number of characters before automatically inserting a new line within a cross-reference list.
Default: `100`

Value Returned

`t`	Created offsheet pin references for multisheet designs.
`nil`	Offsheet pin reference creation was unsuccessful.

Examples

Produces cross-references on pins in a schematic using default options.

```
schComputePinRef( cellview )
```

Produces a cross-reference report file in `design.xref`. Use a space " " or a reference separator instead of the default comma " , ". The pin references also appear in the schematic by default.

```
schComputePinRef( cellview "design.xref" ?display "on" ?reportFile nil ?reportDups
"off" ?sortByDir "off" ?reportFile " " )
```

schCopy

```
schCopy(
     d_fig
     d_destCV
     l_transform
     )
     => d_object / nil
```

Description

Copies the given object to the given destination cellview. The object location and orientation can be specified before it is placed at the destination location by the given transformation argument. The copied figure is first rotated and reflected about the origin as specified by the orientation of the transform, then translated by the offset of the transform.

The destination cellview must be editable. This function copies figures between schematic or symbol cellviews only.

Arguments

d_fig	Figure to copy.
d_destCV	Cellview in which to place the copied object. This argument must be a schematic or symbol cellview.
l_transform	Specifies the relative location, orientation, and optionally magnification of the moved figure, specified as a list of the form: (*l_offset t_orient [n_magnification]*) Where: *l_offset* is the offset from the original position expressed as a list of two floats, the first specifying the distance to move in the x direction and the second the distance in the y direction; for example (10.0:5.0). *t_orient* specifies the orientation of the moved object and is one of R0, R90, R180, R270, MX, MXR90, MY, MYR90. The value must be enclosed in double quotes. *n_magnification* specifies the relative size of the moved object. The default is 1.0 (i.e. the same size as before the move).

Value Returned

d_object	The ID of the new figure.
nil	Copying the object was unsuccessful.

Examples

Creates a copy of `fig1` in the cellview specified by `cv2`. The new figure has the same rotation as `fig1` and is translated by offset 10.0, 5.0, with an `R0` orientation.

```
objId = schCopy( fig1 cv2 list(10.0:5.0 "R0") )
```

schCreateInst

```
schCreateInst(
     d_cvId
     d_masterId
     t_instanceName
     l_origin
     t_orient
     [ n_magnification ]
     )
     => d_inst / nil
```

Description

Creates an instance of the given master cellview in the specified cellview at the given location with the given orientation. You can specify the magnification to set for the instance. Although not fully supported, you can use this property to scale the appearance of an instance.

Arguments

d_cvId	Cellview ID of the editable schematic cellview in which to create the instance.
d_masterId	ID of the instance master cellview. You can access the master ID using several different methods; for example, an explicit call to `dbOpenCellViewByType`.
t_instanceName	Instance name to give the instance; must be enclosed in quotation marks. This argument can be `nil`, a simple name, or a name with a vector expression. When the argument is `nil`, a unique instance name will be generated automatically. When the argument is a simple name or a name with a vector expression, the name must be unique among existing instances in the cellview. If the name has a vector expression—for example, `"<0:3>"` —the expression is used to create an iterated instance.
l_origin	Location to place the instance. The origin of the instance master is used as the reference point.
t_orient	Orientation to give the instance placement; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, `MYR90`
n_magnification	Database magnification value to set on the instance. Default: `1.0`

Value Returned

d_inst	The ID of the instance.
nil	Instance creation was unsuccessful.

Examples

Creates an instance of the `inverter` symbol from the sample library in the specified `cvId`. The instance name is `I23` and the name is placed at 0,0 with an `R0` orientation.

```
symbolId = dbOpenCellViewByType( "sample" "inv" "symbol" "" 'r )
instId = schCreateInst( cvId symbolId "I23" 0:0 "R0" )
```

Creates an iterated instance of the same inverter. The instance rotates 90 degrees before being placed.

```
intId = schCreateInst( cvId symbolId "I24<0:1>" 0:1 "R90" )
```

schCreateInstBox

```
schCreateInstBox(
     d_cvId
     [ l_bBox ]
     )
     => d_id / nil
```

Description

Creates an instance box in the given symbol cellview. This function uses a bounding box you specify or determines a bounding box from the pins and device shapes.

Arguments

d_cvId	Cellview ID of an editable symbol cellview in which to create the instance box.
l_bBox	List specifying the corners of the instance box to create. If not specified, or specified as `nil`, a bounding box created from all the pins and device shapes is used.

Value Returned

d_id	The ID of the created instance box.
`nil`	Instance box creation was unsuccessful.

Examples

Creates an instance box in the `inv` symbol cellview based on the pins and device shapes in the cellview.

```
cvId = dbOpenCellViewByType( "sample" "inv" "symbol" "" 'a )
boxId = schCreateInstBox( cvId )
```

Creates an instance box with the specified `bBox` coordinates.

```
boxId = schCreateInstBox( cvId list(0:0 2:2) )
```

schCreateNetExpression

```
schCreateNetExpression(
     d_cvId
     t_netExpr
     d_glueId
     l_point
     t_justify
     t_orient
     t_fontStyle
     n_fontHeight
     )
     => d_id / nil
```

Description

Creates an inherited connection and the corresponding net expression label. Attaches the given net expression to the given database object. It validates the syntax of the expression and attaches a net expression label to the database object. If the object is a schematic wire, you must run the schematic extractor to create the inherited connection. Before calling this function, you must acquire all the required arguments of the function.

You can programmatically create inherited terminals by explicitly calling `dbCreateTermNetExpr`. A net expression label will not be created. You cannot create inherited signals by explicitly calling `dbCreateSigNetExpr` because the schematic extractor deletes an inherited signal that does not have a net expression label.

Arguments

d_cvId	Cellview ID of the cellview in which to create the expression.
t_netExpr	The net expression in NLP syntax; must be enclosed in quotation marks.
d_glueId	The database object to associate the net expression with. It must be either a schematic wire, schematic pin, or symbol pin object.
l_point	The origin point to locate the net expression.
t_justify	Justification to give the net expression label text with respect to its placement; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`

t_orient	Orientation to give the placement of the net expression; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, `MYR90`
t_fontStyle	Label font style; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`
n_fontHeight	Label height in user units. Default: `0.0625`

Value Returned

d_id	The ID of the net expression label for the inherited connection.
`nil`	There is a syntax error in the given expression or the parent object is not a schematic wire, schematic pin, or symbol pin.

Examples

Creates the net expression label `[@power:%:vdd!]` glued to the specified wire figure at location 0, 1.875. The label is control-point justified at the lower left of the label, the font is a fixed-width font, and the height is 0.125 user units.

```
netExprLabelId = schCreateNetExpression( cv "[@power:%:vdd!]" wireId (0:1.875)
"lowerLeft" "R0" "fixed" 0.125 )
```

Related Topics

Syntax of an Inherited Net Expression

schCreateNoteLabel

```
schCreateNoteLabel(
     d_cvId
     l_point
     t_text
     t_just
     t_orient
     t_fontStyle
     n_fontHeight
     t_type
     )
     => d_label / nil
```

Description

Creates note labels in a schematic or symbol cellview with the attributes and properties you specify. These labels do not affect the connectivity but can be useful for annotation.

Arguments

d_cvId	Cellview ID of an editable schematic or symbol cellview in which to create the note label.
l_point	Location of the note label specified as a point.
t_text	Text of the note label; must be enclosed in quotation marks.
t_just	Justification of the label text with respect to its placement. Use string values; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
t_orient	Orientation of the note label; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, `MXR90`
t_fontStyle	Font style of the label; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`
n_fontHeight	Label height in user units. Default: `0.0625`
t_type	Type of label to create; must be enclosed in quotation marks. Valid Values: `normalLabel`, `NLPLabel`, `ILLabel`

Value Returned

d_label	The ID of the new label.
nil	Note label creation was unsuccessful.

Example

Creates a note label called any Text String in the specified cellview located at 0,1.875 with no rotation. The label's control point is justified at the lower left of the label, the font is a fixed-width font, the height is 0.125 user units, and it is a normal label.

```
labelId = schCreateNoteLabel( cv 0.0:1.875 "any Text String" "lowerLeft" "R0"
"fixed" 0.125 "normalLabel" )
```

schCreateNoteShape

```
schCreateNoteShape(
     d_cvId
     t_type
     t_lineStyle
     l_points
     [ n_width ]
     )
     => d_shape / nil
```

Description

Creates note shapes in a schematic or symbol cellview with the attributes and properties you specify. These shapes do not affect the connectivity but can be useful for annotation.

Arguments

d_cvId	Cellview ID of an editable schematic or symbol cellview in which to create the note shape.
t_type	Type of shape to create; must be enclosed in quotation marks. Valid Values: `line`, `rectangle`, `polygon`, `arc`, `ellipse`, `circle`
t_lineStyle	Line style of the shape; must be enclosed in quotation marks. Valid Values: `solid`, `dashed`
l_points	Location of the note shape specified as a list of at least two points.
n_width	Width of the line.

Value Returned

d_shape	The ID of the new shape.
`nil`	Note shape creation was unsuccessful.

Examples

Creates a rectangle in the specified cellview whose lower left corner is at 0,0 and upper right corner is at 10,10. The boundary of the rectangle is displayed as a solid line.

```
shapeId = schCreateNoteShape( cv "rectangle" "solid" list(0:0 10:10) )
```

schCreatePin

```
schCreatePin(
     d_cvId
     d_master
     t_termName
     t_direction
     g_offSheetP
     l_origin
     t_orientation
     [ g_powerSens ]
     [ g_groundSens ]
     [ g_sigType ]
     )
     => d_pin / nil
```

Description

Creates instances that are used to represent pins of terminals in a schematic cellview. Creates only a pin in a schematic cellview. The destination cellview must not be the same as the master cellview and must be editable.

Arguments

d_cvId	Cellview ID of an editable schematic cellview in which to create the pin.
d_master	Master cellview to which the pin instance refers. If you pass `nil`, the pin master is selected from the pin list based on the arguments *t_direction* and *g_offSheetP*.
t_termName	Terminal name created for the pin; must be enclosed in quotation marks.
t_direction	I/O direction of the pin terminal; must be enclosed in quotation marks. Valid Values: `input`, `output`, `inputOutput`, `switch`
g_offSheetP	Specifies whether the pin is an offsheet connector. Valid Values: `t`, `nil`
l_origin	Origin of the pin specified as a point.
t_orientation	Orientation of the pin relative to its placement; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MY`, `MYR90`, `MX`, `MXR90`

g_powerSens	Name of the power terminal in d_cvId to which the pin is sensitive, or nil if not required.
g_groundSens	Name of the ground terminal in d_cvId to which the pin is sensitive, or nil if not required.
g_sigType	Type of signal carried by the pin. If you omit this argument, or pass nil, then the pin takes the signal type of an existing wire with the same name, or "signal" if there is no such wire. Valid Values: `analog`, `clock`, `ground`, `power`, `reset`, `scan`, `signal`, `tieHi`, `tieLo`, `tieOff`, `nil`

Value Returned

d_pin	The ID of the new pin.
`nil`	Instance creation was unsuccessful.

Examples

Creates a pin in the specified cellview. The pin is created from the `inputCVId` master cellview and assigned `I1` pin name with `input` direction. The pin is not an offsheet pin and is placed at 0,0 with no rotation.

```
inputCVId = dbOpenCellViewByType( "basic" "ipin" "symbol" "" 'r )
pinId = schCreatePin( cvId inputCVId "I1" "input" nil 0:0 "R0" )
```

schCreateSheet

```
schCreateSheet(
     d_cvId
     x_number
     t_borderLibrary
     t_borderCell
     t_borderView
     )
     => d_sheetInstId / nil
```

Description

Creates a new sheet for a multisheet schematic.

The schematic is generated based on the cell name of the index with the sheet number appended; for example, `sheet003`. A multisheet symbol is created with the `msymbol` view and an instance is placed in the index schematic.

If the source is not a multisheet schematic, it is converted into a multisheet schematic. An index is created, and the source becomes the specified sheet number in the multisheet design.

If the numbered sheet already exists, the new sheet is inserted before the existing sheet. The remaining sheets are renumbered. Also, if the sheet number is less than or equal to zero, a sheet number is generated based on the value of the last sheet number in the multisheet schematic.

When you specify the library, cell, and view of a sheet border master, a border instance is added to the new multisheet schematic.

Arguments

d_cvId	Cellview ID of the source editable schematic, which must be an index schematic or a nonsheet schematic cellview.
x_number	Number of the new sheet.

t_borderLibrary	Name of the library containing the sheet border master. If specified as an empty string or `nil`, the new sheet is created without a sheet border. Also used to specify the library containing the `sheetSymbol` to use when creating the sheet instance in the index schematic that represents the new sheet. If specified as an empty string or `nil`, each library in your `cds.lib` file is searched until a `sheetSymbol` cell that has an `msymbol` view is found.
t_borderCell	Cell name of the sheet border master. If specified as an empty string or as `nil`, the new sheet is created without a sheet border; must be enclosed in quotation marks.
t_borderView	View name of the sheet border master. If specified as an empty string or as `nil`, the new sheet is created without a sheet border; must be enclosed in quotation marks.

Value Returned

d_sheetInstId	The instance ID of the new sheet instance in the index schematic.
`nil`	Sheet creation was unsuccessful.

Examples

Creates sheet number `4`, with an A-sized sheet border from the `US_8ths` library in the multisheet schematic defined by the given `index` schematic.

```
sheetInstId = schCreateSheet( indexId 4 "US_8ths" "Asize" "symbol" )
```

Converts an ordinary schematic into sheet `1` of a multisheet schematic and creates an index schematic with the same name as the original schematic. Searches each library specified in your `cds.lib` file until a `sheetSymbol` is found to create an instance representing the new sheet in the `index` schematic.

```
sheetInstId = schCreateSheet( cvId 1 "" "" "" )
```

Converts an ordinary schematic into sheet `2` of a multisheet schematic and creates an index schematic with the same name as the original schematic. A D-sized border is added to the converted schematic.

```
sheetInstId = schCreateSheet( cvId 2 "US_8ths" "Dsize" "symbol" )
```

Creates sheet number 4, without a sheet border, in the multisheet schematic defined by the given index schematic. Uses the `sheetSymbol` from the `US_8ths` library to create an instance representing the new sheet in the index schematic.

```
sheetInstId = schCreateSheet( indexID 4 "US_8th" "" "")
```

schCreateSplitPrimarySymbol

```
schCreateSplitPrimarySymbol(
     d_cellId
     )
     => t / nil
```

Description

Searches for split symbol views within the given cell and creates a split-primary symbol by assembling all the terminals from the gathered views. The generated split-primary cellview has the view name as `symbol`. This function removes an existing split-primary cellview, if any, and replaces it with a new one.

Arguments

d_cellId	Specifies the ID of the cell for which the split-primary symbol is to be generated.

Value Returned

`t`	The split-primary symbol has been successfully created.
`nil`	Returns `nil` if: ■ the cell directory is read-only. ■ the symbol view directory is read-only. ■ there exists a user-defined view by the name `symbol`. ■ the `symbol` view is locked by another process when the specified cellview remains unchanged.

Examples

The following example creates a split-primary symbol, `bga`, and saves it in a library, `lib`.

```
bga = ddGetObj("lib" "bga")
schCreateSplitPrimarySymbol(bga)
```

schCreateSymbolLabel

```
schCreateSymbolLabel(
     d_cvId
     l_point
     t_labelChoice
     t_text
     t_justify
     t_orient
     t_fontStyle
     n_fontHeight
     t_type
     )
     => d_label / nil
```

Description

Creates a label in only a symbol cellview with the specified attributes that is opened in append mode.

Arguments

d_cvId	Cellview ID of a symbol cellview in which to create the label.
l_point	Location of the label specified as a point.
t_labelChoice	Type of label to create; must be enclosed in quotation marks. Valid Values: `instance label`, `device label`, `device annotate`, `pin name`, `pin annotate`
t_text	Text of the label; must be enclosed in quotation marks.
t_justify	Justification of the label text with respect to its placement; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
t_orient	Orientation of the instance placement; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, `MYR90`
t_fontStyle	Font style of the label; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`

n_fontHeight	Label height in user units. Default: `0.0625`
t_type	Type of label to create; must be enclosed in quotation marks. Valid Values: `normalLabel`, `NLPLabel`, `ILLabel`

Value Returned

d_label	The ID of the new label.
`nil`	Label creation was unsuccessful.

Examples

Creates an instance label `[@instanceName]` in the specified cellview, located at 0,1.875. The label's control point is justified at the lower left of the label, the font is a fixed-width font, the height is 0.125 user units, and the label is an interpreted `NLPLabel` label.

```
labelId = schCreateSymbolLabel( cv 0:1.875 "instance label" "[@instanceName]"
"lowerLeft" "R0" "fixed" 0.125 "NLPLabel" )
```

schCreateSymbolPin

```
schCreateSymbolPin(
     d_cvId
     d_master
     t_termName
     t_direction
     l_origin
     t_orientation
     [ g_flatten ]
     [ g_powerSens ]
     [ g_groundSens ]
     [ g_sigType ]
     )
     => t_pinFigId / nil
```

Description

Creates a pin in the given cellview with the name, direction, and orientation you specify.

The figures that describe the pin are taken from the given pin master cellview, which can be accessed with a call to `dbOpenCellViewByType`, and are copied into the specified cellview. A terminal is created for the pin with the given name. The objects are created in the cellview with the specified orientation.

Arguments

d_cvId	Cellview ID of an editable symbol cellview in which to create the pin.
d_master	Master cellview containing the objects that specify the symbol pin.
t_termName	Name for the terminal that is created for the pin; must be enclosed in quotation marks.
t_direction	I/O direction of the pin terminal; must be enclosed in quotation marks. Valid Values: `input`, `output`, `inputOutput`, `switch`
l_origin	Location for the pin specified as a point.
t_orientation	Orientation of the pin placement; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, `MYR90`

g_flatten	Controls whether the pin figures are copied from *d_master* into *d_cvId* (that is, flattened) or placed as instances. When set to nil, the pin is created by making an instance of *d_master* in *d_cvId*. Otherwise, the pin is created by copying the pin figures from *d_master* into *d_cvId*. The default value is t.
g_powerSens	Name of the power terminal in *d_cvId* to which the pin is sensitive, or nil if not required.
g_groundSens	Name of the ground terminal in *d_cvId* to which the pin is sensitive, or nil if not required.
g_sigType	Type of signal carried by the pin. If you omit this argument, or pass nil, then the pin takes the signal type of an existing wire with the same name, or "signal" if there is no such wire. Valid Values: analog, clock, ground, power, reset, scan, signal, tieHi, tieLo, tieOff, nil

Value Returned

t_pinFigId	The ID of the new pin figure.
nil	Pin creation was unsuccessful.

Examples

Creates terminal A, takes the objects from the symbol pin master masterCV, and creates corresponding objects in the symbolCV cellview. The objects are placed relative to the 0,0 location without rotation.

```
symPinId = schCreateSymbolPin( symbolCV masterCV "A" "input" 0:0 "R0" )
```

schCreateSymbolShape

```
schCreateSymbolShape(
     d_cvId
     t_shape
     t_style
     l_points
     [ n_width ]
     )
     => d_shapeId / nil
```

Description

Creates the specified shape using the given fill style and the list of points in the given cellview.

Arguments

d_cvId	Cellview ID of an editable symbol cellview in which to create the shape.
t_shape	Type of shape to create; must be enclosed in quotation marks. Valid Values: `line`, `rectangle`, `polygon`, `arc`, `circle`, `ellipse`
t_style	Fill style of the shape to create; must be enclosed in quotation marks. Valid Values: `outline`, `solid`
l_points	List of points for the specified shape.
n_width	Width of the line.

Value Returned

d_shapeId	The ID of the specified shape.
`nil`	Shape creation was unsuccessful.

Examples

Creates a solid rectangular shape between points 0:0 and 1:1.

```
shapeId = schCreateSymbolShape( cv "rectangle" "solid" list(0:0 1:1) )
```

schCreateWire

```
schCreateWire(
     d_cvId
     t_entryMethod
     t_routeMethod
     l_points
     n_xSpacing
     n_ySpacing
     n_width
     [ t_color ]
     [ t_lineStyle ]
     )
     => l_wireId
```

Description

Creates flight lines, wide wires, or narrow wires in the specified schematic cellview.

Arguments

d_cvId	Cellview ID of a schematic cellview in which to create the wire.
t_entryMethod	Wire entry method; must be enclosed in quotation marks. If you specify *t_entryMethod* as `draw`, the resulting wires are created using the given list of points and *t_routeMethod* is ignored. If you specify *t_entryMethod* as `route`, *t_routeMethod* is applied and only the first two points in the list of points are used. Valid Values: `draw`, `route`
t_routeMethod	Method to use when routing the wires; must be enclosed in quotation marks. This argument applies only when *t_entryMethod* is `route`. If you specify *t_routeMethod* as `flight`, flight lines are created between the points specified. If you specify *t_routeMethod* as `direct` or `full`, the appropriate routing algorithm is applied to route the wires between the points. Default: `flight`, `direct`, `full`
l_points	List of points to use to create the wire. This can be any number of points, but the system creates as many two-point line segments as needed to exhaust the list of points.
n_xSpacing	Horizontal snap spacing to apply to the specified point.

n_ySpacing	Vertical snap spacing to apply to the specified point.
n_width	Physical width of the wire. A width of `0` specifies a line. A width greater than `0` specifies a wide wire.
t_color	The color of the wire. The color must be defined in the Display Resource File. If *t_routeMethod* is `flight`, *t_color* is ignored.
t_lineStyle	The line style of the wire. The line style must be defined in the Display Resource File. If *t_routeMethod* is `flight`, *t_lineStyle* is ignored.

Value Returned

l_wireId	A list of database objects for each wire segment you create.

Examples

Creates a wire from 0:0 to 1:0.

```
schCreateWire( cv "draw" "full" list(0:0 1:0) 0.0625 0.0625 0.0 )
```

Routes a wide wire from 0,0 to 1,20.

```
schCreateWire( cv "route" "full" list(0:0 1:20) 0.0625 0.0625 0.05 )
```

schCreateWireLabel

```
schCreateWireLabel(
     d_cvId
     d_glue
     l_point
     t_text
     t_justify
     t_orient
     t_fontStyle
     n_fontHeight
     g_aliasP
     )
     => d_labelId / nil
```

Description

Creates wire labels and glues them to the object you specify.

Arguments

d_cvId	Cellview ID of an editable schematic cellview in which to create the wire label.
d_glue	Wire or pin on which to glue the label.
l_point	Location of the label specified as a point.
t_text	Text of the label.
t_justify	Justification of the label text with respect to its placement; must be enclosed in quotation marks. Valid Values: `upperLeft`, `upperCenter`, `upperRight`, `centerLeft`, `centerCenter`, `centerRight`, `lowerLeft`, `lowerCenter`, `lowerRight`
t_orient	Orientation of the label; must be enclosed in quotation marks. Valid Values: `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, `MYR90`
t_fontStyle	Font style of the label; must be enclosed in quotation marks. Valid Values: `euroStyle`, `fixed`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`, `milSpec`
n_fontHeight	Label height in user units.
g_aliasP	Label alias flag, which specifies if a wire label has an alias or a normal net name. Valid Values: `t`, `nil`

Value Returned

d_labelId	The ID of the new wire label.
nil	Unsuccessful.

Examples

Creates the wire label net1 glued to the specified wire figure at location 0,1.875. The label is control-point justified at the lower left of the label, the font is a fixed-width font, and the height is 0.1 user units.

```
schCreateWireLabel( cv wireId (0:1.875) "net1" "lowerLeft" "R0" "fixed" 0.1 nil )
```

schDelete

```
schDelete(
     d_fig
     )
     => t / nil
```

Description

Deletes the figure or object you specify only from schematic or symbol cellviews.

Arguments

d_fig	Figure to delete.

Value Returned

`t`	Deleted the figure or object you specify only from schematic or symbol cellviews.
`nil`	Deletion was unsuccessful.

Examples

Deletes `fig1`.

```
schDelete( fig1 )
```

schDeleteIndex

```
schDeleteIndex(
     d_cvId
     )
     => t / nil
```

Description

Deletes an index schematic if there is one remaining sheet. Converts the remaining sheet into an ordinary schematic with the cell name of the index schematic and replaces any offsheet pins with schematic pins.

Arguments

d_cvId	Cellview ID of the index schematic to delete.

Value Returned

t	Deleted an index schematic if there is one remaining sheet.
nil	Deletion was unsuccessful.

Examples

Deletes the multisheet index and converts the remaining sheet into an ordinary schematic and converts any offsheet pins to schematic pins.

```
schDeleteIndex( cv )
```

schDeleteSheet

```
schDeleteSheet(
     d_cvId
     x_number
     )
     => t / nil
```

Description

Deletes a sheet from a multisheet schematic design.

Arguments

d_cvId	Cellview ID of the index schematic.
x_number	Number of the sheet to delete.

Value Returned

t	Deleted a sheet from a multisheet schematic design.
nil	Deletion was unsuccessful.

Examples

Deletes sheet number 3 from the multisheet index schematic.

```
schDeleteSheet( cv 3 )
```

schDeselectAllFig

```
schDeselectAllFig(
     [ d_cvId ]
     )
     => t
```

Description

Deselects all objects in a specified cellview. Bypasses the selection filter.

Arguments

d_cvId	Cellview ID of the cellview containing the objects you want to deselect. If no cellview is specified, the current cellview is used.

Value Returned

t	Deselects all objects in a specified cellview.

Examples

Deselects all figures from the cellview in the current window.

```
schDeselectAllFig()
```

schDistribute

```
schDistribute(
     l_list
     s_justify
     )
     => t / nil
```

Description

Arranges objects at equal distance in the specified direction in the schematic or symbol view.

Arguments

l_list	List of database objects to be distributed.
s_justify	Direction of arranging objects with equal space, such as horizontal or vertical.

Value Returned

t	Distribution was done successfully.
nil	Distribution was unsuccessful.

Examples

```
schDistribute( geGetSelSet() 'vertical )

schDistribute( geGetSelSet() 'horizontal )
```

Related Topics

Distributing

schDrawSymbolPin

```
schDrawSymbolPin(
     d_cvId
     t_termName
     t_direction
     l_points
     )
     => t_pinFigId / nil
```

Description

Creates a symbol pin in the specified cellview by creating a terminal of the given name with the given direction and a polygon shape specified by the given list of points.

Can be used only when editing a symbol. *d_cvId* must be editable.

Arguments

d_cvId	Cellview ID of an editable symbol cellview ID in which to create the pin.
t_termName	Name for the terminal that is created for the pin; must be enclosed in quotation marks.
t_direction	I/O direction for the pin terminal; must be enclosed in quotation marks. Valid Values: `input`, `output`, `inputOutput`, `switch`
l_points	List of points that specify the shape of the polygon that represents the pin.

Value Returned

t_pinFigId	The ID of the new pin shape.
`nil`	Symbol creation was unsuccessful.

Examples

Creates a pin with terminal `A` with `input` direction using a shape specified by four points.

```
pinFigId = schDrawSymbolPin( cvId "A" "input" list(0:0 0.0625:0 0.0625:0.0625
0:0.0625) )
```

schEditPinOrder

```
schEditPinOrder(
     d_cvId
     l_pinList
     g_updateInstLastChanged
     )
     => t / nil
```

Description

Updates the pin ordering for schematic or symbol cellviews given a list of pin names contained in the cellview and their desired order.

The purpose of this function is to keep the pin ordering of a schematic or symbol synchronized with the port ordering of a Verilog® HDL or VHDL model.

Arguments

d_cvId	Cellview ID of an editable schematic or symbol cellview.
l_pinList	List of ordered pin names.
g_updateInstLastChanged	Boolean flag specifying whether to update the time stamp for the instances last changed.

Value Returned

t	Updated the pin ordering for schematic or symbol cellviews given a list of pin names contained in the cellview and their desired order.
nil	Pin ordering update was unsuccessful.

Examples

Sets the pin order for the cellview ID to q, qbar, d, clk, and preset.

```
pinList = list( "q" "qbar" "d" "clk" "preset" )
schEditPinOrder( cvId pinList nil )
```

schEditSheetSize

```
schEditSheetSize(
     d_cvId
     t_borderLib
     t_borderCell
     t_borderView
     )
     => t / nil
```

Description

Places or replaces a sheet border instance in a schematic. This function works for both multisheet and non-multisheet schematics.

Arguments

d_cvId	Cellview ID of an editable schematic to modify.
t_borderLib	Name of the library containing the sheet border master; must be enclosed in quotation marks. Use an empty string if you want no border.
t_borderCell	Cell name of the sheet border master; must be enclosed in quotation marks. Use an empty string if you want no border.
t_borderView	View name of the sheet border master; must be enclosed in quotation marks. Use an empty string if you want no border.

Value Returned

t	Placed or replaced a sheet border instance in a schematic.
nil	Sheet border instance placement or replacement was unsuccessful.

Examples

Adds an A-sized sheet border to the schematic you specify. If the schematic already contains a sheet border, it is replaced with the A-sized sheet border.

```
schEditSheetSize( cv "US_8ths" "Asize" "symbol" )
```

Removes any existing sheet borders.

```
schEditSheetSize( cv "" "" "" )
```

schExportSVG

```
schExportSVG(
     ?windowId d_windowId
     ?cellView d_cellview
     ?path s_path
     ?background t_background
     ?bgColor t_bgColor
     ?foreground t_foreground
     ?fgColor s_fgColor
     ?grid S_grid
     ?stipple S_stipple
     ?color S_color
     ?colorType t_colorType
     ?drawBBox g_drawBBox
     ?nameCallback g_nameCallback
     )
     => t / nil
```

Description

Exports a schematic as a vector image in SVG format.

Arguments

`?windowId` *`d_windowId`*

The window which to export from or `nil` a cellview must be passed.
Default: `nil`

`?cellView` *`d_cellView`*

A valid schematic cellview which is to be exported.
Default: `none`

`?path` *`s_path`*

Path to a file to be exported (`"path/test.svg"`) or `nil`.
Default: `nil`

`?background` *`t_background`*

Background color to be used when exporting. If not specified, and a `windowId` is specified, the background color follows that of the specified window. Otherwise, the background color is white.
Default: `white`

`?bgColor` *`t_bgColor`*

Alias of the *background* argument. Ignored if *background* is specified.

Note: This argument provides compatibility with hiExportImage, which uses ?bgColor to specify the foreground color. In other cases, it is recommended to use the ?background argument.

?foreground *t_foreground*

The color to use as the foreground color for bicolor and monochrome exports. If the background color is not specified, the foreground color is black.
Default: black

?fgColor *t_fgColor*

Alias of the *foreground* argument. Ignored if *foreground* is specified.

Note: This argument provides compatibility with hiExportImage, which uses ?fgColor to specify the foreground color. In other cases, it is recommended to use the ?foreground argument.

?grid *S_grid* Grid style to be used.
Valid values: none, dotted, line
Default: none

?stipple *S_stipple* Stippling style to be used.
Valid values: alpha, bitmap, vector
Default: alpha

?color *S_color* Color mapping mode.

- adjustForeground: Adjusts canvas colors to contrast effectively with the background color. Default setting when left unspecified.
- fullColor: Uses the canvas colors directly.
- hiMonochrome: Interpolates canvas colors between the background color and the foreground color the same way as hiExportImage.
- hiBiColor: Draws everything with the foreground color

Default: adjustForeground

?colorType *t_colorType*

Selects a color mapping mode based on `hiExportImage`. Valid values: `fullColor`, `monochrome`, `biColor`

`?drawBBox` *`g_drawBBox`*

Controls whether to draw instance selection boxes. Default: `t`

`?nameCallback` *`g_nameCallback`*

Lets you pass in a callable which is used to name the instance selection boxes. You can also use `nil`. Example: `(lambda (fig path) fig->name)`

Value Returned

When the `?path` is specified:

`t`	SVG file successfully saved to disk.
`nil`	SVG export failed.

When the `?path` is `nil`:

`string`	String containing the SVG text.
`nil`	SVG export failed.

Examples

The current window is exported to the file `"test.svg"`.

```
when( window = hiGetCurrentWindow()
        schExportSVG(?windowId window ?path "test.svg")
    )
```

You get a string from the SVG exporter which you can then write to disk or insert into another file or export.

```
when(window = hiGetCurrentWindow()
        schExportSVG(?windowId window
                     ?background "black"
                     ?stipple 'vector)
    )
    when(svg
```

```
        println(svg)
    )
```

schExtendSelSet

```
schExtendSelSet(
     w_windowId
     l_pt
     )
     => t / nil
```

Description

Extends the selection of the object in the specified position by selecting the object around the current object.

Searches through the schematic cellview for objects that touch the object in the specified position and adds them to the selected set. For example, extending a wire selects all segments in the same branch (stopping at T-intersections, pins, instance pins, or changes in wire width). The function extends it again and selects all objects in the path, stopping only at pins and instance pins.

You can extend an instance to select all wire segments connected to any of its instance pins. Repetitive extended selection of an instance extends the wires as defined above.

You can extend labels to apply to more objects. Repetitive extended selection of a label extends the label as defined above.

When this function reaches the maximum selection level, it cycles back to the single object.

Arguments

w_windowId	Window to which to apply the selection.
l_pt	Point that specifies the location of the selection.

Value Returned

t	Extends the selection of the object in the specified position by selecting the object around the current object.
nil	Selection extension was unsuccessful.

Examples

If the specified point is over the object, this function selects the object. If the object is already selected, this function extends the object. Any objects in the next selection level are added to the selected set. You can incrementally increase the selection level until an object is selected. If the function reaches the maximum extension level, it cycles back to a single object.

```
schExtendSelSet( hiGetCurrentWindow( ) hiGetCommandPoint( ) )
```

schExtractConn

```
schExtractConn(
     d_cvId
     )
     => l_result
```

Description

Runs the schematic connectivity extractor on the cellview you specify.

Figures on the wire layer with `drawing`, `flight`, or `label` purposes are processed. Figures on the pin layer with `drawing` purposes are processed as schematic pins. Instances are of either `cell` or `pin` purpose; components that have `cell` purpose and `pin` instances must have objects in the master on the `pin` layer with `drawing` purpose to be processed correctly.

The extractor uses three schematic environment settings:

- `maxLabelOffsetUU` specifies an offset distance from a label in which automatic association, or gluing, occurs

  Refer to schGlueLabel for details. If a wire is within the distance specified by `maxLabelOffsetUU`, the label is automatically glued to it.

- `runSRC` specifies whether the schematic rules checker is run after the connectivity is successfully extracted from the cellview

- `runVIC` specifies whether the cross-view checker is run after the connectivity is successfully extracted from the cellview

Note: You can run the function `schClearConn` to remove existing schematic connectivity on the cellview before restarting the extraction using `schExtractConn`.

If you initiate the extraction from the index of a multisheet design, the extractor automatically extracts the sheets that require extracting.

Can be used only when editing a schematic cellview.

Note: Cadence recommends that you use `schCheck` instead of this function and that you replace existing calls to this function with calls to `schCheck`.

Arguments

d_cvId	Cellview ID of the cellview from which to extract connectivity.

Value Returned

l_result	A list containing the errors and total number of warnings generated, in that order. This also includes errors or warnings from the schematic rules checker or cross-view checker.

Examples

Extracts connectivity for the cellview you specify.

```
schExtractConn( cv )
```

Extracts connectivity for the cellview you specify but does not run the schematic rules checker or the cross-view checker.

```
schSetEnv( "runSRC" nil )
schSetEnv( "runVIC" nil )
result = schExtractConn( cv )
```

Related Topics

schGlueLabel

schClearConn

schCheck

schExtractStatus

```
schExtractStatus(
     d_cvId
     )
     => t_status / nil
```

Description

Checks for error and warning markers before returning the schematic extraction status for the specified schematic cellview. When the schematic is read-only, schExtractStatus() will also check whether any instance masters have been updated since the schematic was last saved.

Arguments

d_cvId	Cellview ID of the schematic cellview to check.

Value Returned

t_status	The string `obsolete` if the cellview has been updated since the last time the connectivity was extracted for the cellview, `dirty` if the connectivity is current but there are error or warning markers in the cellview, and `clean` if the connectivity is current and there are no error or warning markers in the cellview.
`nil`	Check for error and warning markers was unsuccessful.

Examples

```
cvId = dbOpenCellViewByType( "lib" "block" "schematic" "" 'r )
case( schExtractStatus( cvId )
    (obsolete info("Re-Check schematic.\n"))
    (dirty info("Ok but look it over.\n"))
    (clean info("GO FOR IT!\n"))
    )
)
```

schFindIgnorePropByName

```
schFindIgnorePropByName(
     t_name
     )
     => l_list / nil
```

Description

Searches for an ignore property in the registered property set. You can view the registered ignore property set by clicking *Options – Editor – Ignore Properties* tab in the schematic window.

Arguments

t_name	Name of the property to be searched in the ignore property set.

Value Returned

l_list	A disembodied property list that contains all the information about the searched ignore property, such as property name, prompt name, type, value, and a flag to determine whether a particular ignore property is enabled or not.
`nil`	The specified property is not found in the ignore property set.

Examples

```
schFindIgnorePropByName( "nlIgnore" )
```

It returns the following list if this property exists in the registered properties:

```
(nil name "nlIgnore" prompt "nlIgnore"
type "string" value "spectre" enabled
nil
)
```

If the searched property does not exists in the registered ignore properties, it returns `nil`.

```
schFindIgnorePropByName( "nlAction" ) ==> nil
```

schFlatten

```
schFlatten(
     d_inst
     [ ?at g_at ]
     )
     => l_list / nil
```

Description

Flattens the contents of the schematic cellview under the passed instance into the cellview of the instance. To control which underlying schematic is picked, configure `viewList` first.

Arguments

d_inst	The ID of the instance which has the schematic cellview to be flattened.
`?at` *g_at*	The location where the flattened content is placed.

Value Returned

l_list	A disembodied property list that contains the names of the instances and all the renames that have been performed.
`nil`	The contents of the schematic cellview could not be flattened.

Examples

Finds an instance in the schematic and flattens it, auto-placing the contents.

```
inst = dbFindAnyInstByName(cv "I0")
schFlatten(inst)
```

Determines where to place the flattened content. The bottom left corner of the content is placed at the specified coordinate.

```
inst = dbFindAnyInstByName(cv "I1")
schFlatten(inst ?at 0:3)
```

Flattens `schematic2` if it exists. If it does not exist, it tries to flatten `schematic` next, and so on.

```
schSetEnv("viewList" "schematic2 schematic symbol")
inst = dbFindAnyInstByName(cv "I2")
```

```
schFlatten(inst ?at 0:3)
```

schGetAllIgnoreProps

```
schGetAllIgnoreProps(
     )
     => l_list / nil
```

Description

Returns a disembodied property list for all the ignore properties that are currently registered. You can view the set of registered ignore properties set on the *Ignore Properties* tab by clicking *Options – Editor* in the schematic window.

Arguments

None

Value Returned

l_list	A disembodied property list that contains all the information about the ignore properties, such as property name, prompt name, type, value, and a flag to determine whether a particular ignore property is enabled or not.

Examples

```
schGetAllIgnoreProps()
```

Returns the following list:

```
((nil name "ignore" prompt "ignore"
    type "boolean" value "TRUE" enabled
    nil
    )
    (nil name "lvsIgnore" prompt "lvsIgnore"
    type "boolean" value "TRUE" enabled
    nil
    )
    (nil name "nlIgnore" prompt "nlIgnore"
    type "string" value "spectre" enabled
    nil
    )
)
```

schGetBundleDisplayMode

```
schGetBundleDisplayMode(
     d_labelId
     )
     => vertical / horizontal / nil
```

Description

Takes a label Id and returns its display mode. This only works for label type objects. If the specified object Id is not a label object, then the API will return `nil`. This will only work when the cellview is editable.

Arguments

d_labelId	Label Id whose bundle display mode value, horizontal or vertical, is to be obtained.

Value Returned

`vertical`	Label Id display mode is set to vertical.
`horizontal`	Label Id display mode is set to horizontal.
`nil`	Object Id entered was not a label.

Examples

If `labId` represents a wire bundle label, which is being displayed vertically, then:

```
schGetBundleDisplayMode (labId) => "vertical"
```

If `figId` represents the Id of a pin name, then:

```
schGetBundleDisplayMode (figId) => nil
```

schGetCellViewListInSearchScope

```
schGetCellViewListInSearchScope(
     d_cvId
     t_scope
     d_topCV
     t_viewNameList
     t_libName
     t_mode
     )
     => l_cvList / nil
```

Description

Returns a list of cellviews in the search scope you specify. Only cellviews of the same view type as the base cellview are returned.

Note: To ensure the cellviews returned are correctly released, call `dbClose()` on each cellview after use. Refer to the example below for details.

Arguments

d_cvId	Cellview ID of the schematic or symbol cellview in which to base the search. *d_cvId* must be a schematic or symbol cellview. Hierarchy is not supported for symbol cellviews.
t_scope	Scope of the search; must be enclosed in quotation marks. Valid Values: `selected set`, `cellview`, `hierarchy`, `library`
d_topCV	ID of the top-level cellview from which to start the hierarchical search. This argument is used only when *t_scope* is `hierarchy`.
t_viewNameList	A string of view names that specify the expansion of the hierarchy. This argument is used only when *t_scope* is `hierarchy`.
t_libName	Name of the library in which to search. This argument is used only when *t_scope* is `library`.
t_mode	Access mode used to open the cellviews found during the search; must be enclosed in quotation marks. Valid Values: `read`, `write`

Value Returned

l_cvList	The list is a list of “writable” and “readable” cellviews. If *t_mode* is `write`, the system tries to get write access for each cellview found. If it can get write access, it puts the cellview in the writable list; otherwise, it puts it in the readable list. If *t_mode* is `read`, the system puts all the cellviews in the readable list.
`nil`	Returning the list was unsuccessful.

Examples

Returns a list of schematic cellviews that are in the hierarchy underneath the cellview `top`. The cellviews are placed in the readable list (second element).

```
cvId = dbOpenCellViewByType( "projectLib" "top" "schematic" "" 'r )
retList = schGetCellViewListInSearchScope( cvId "hierarchy" cvId "schematic
   symbol" "" "read" )
cvList = cadr( retList )

; use the cellviews in cvList here before freeing them
foreach( cvInfo cvList dbClose(cvInfo) )
```

schGetCheckGroups

```
schGetCheckGroups(
     )
     => list / nil
```

Description

Returns an association list of all the custom schematic checker groups created using schRegisterCheckGroup().

Arguments

None

Return Value

List	A list of `schRegisterCheckGroup` struct objects in the order that they were initially registered
Nil	If `schRegisterCheckGroup` has not yet been called.

Examples

```
schGetCheckGroups()

=> ((ercChecks schCheckGroup@0x1155e788))
```

Related Topics

`schRegisterCheckGroup`

schGetEnv

```
schGetEnv(
     t_variableName
     )
     => g_value
```

Description

Gets the value of a schematic environment variable.

Along with the `schSetEnv` function, this function lets you program the values for various options within the schematic editor without using a form. Also, these functions complement the general environment variable mechanism, which lets you preset values at startup using a `.cdsenv` file.

Arguments

t_variableName	Name of the schematic environment variable whose value you want to get; must be enclosed in quotation marks.

Value Returned

g_value	Current value of the specified variable.

Examples

Returns the value of the `maxLabelOffsetUU` environment variable.

```
result = schGetEnv( "maxLabelOffsetUU" )
```

Related Topics

schSetEnv

schGetIgnoredStatus

```
schGetIgnoredStatus(
     d_instId
     )
     => l_list
```

Description

Returns the ignored status of the given instance ID. This function can be used only when the instance IDs are known.

Arguments

d_instId	ID of the instance to be checked for ignored status.

Value Returned

l_list	Returns the ignored status. Its value can be one of the following: ■ `invalid`: for all non-instance IDs or invalid instance IDs. ■ `not ignored`: the instance does not have any of the ignore properties enabled. ■ `ignored`: the instance has all the ignore properties enabled. ■ `partially ignored`: The Instance has some of the ignore properties enabled.

Examples

```
schGetIgnoredStatus( car(geGetSelSet()) )
==> ignored
```

schGetMatchingObjects

```
schGetMatchingObjects(
     d_cvId
     t_propName
     t_condOp
     t_propValue
     g_useSelSet
     )
     => l_objects / nil
```

Description

Finds the set of objects that match the specified search criteria in a cellview. You can search by property to limit the search in the selected set.

Arguments

d_cvId	Cellview ID of a schematic or symbol cellview in which to place copied graphics.
t_propName	Property name to search for.
t_condOp	Conditional operator to use during the matching; must be enclosed in quotation marks. Valid Values: ==, !=, <, >, <=, >=
t_propValue	Property value to search for; must be enclosed in quotation marks. If *t_propName* is master, *t_propValue* must be *t_libName t_cellName t_viewName* (separated by spaces).
g_useSelSet	Search is limited to the selected set if set to t; search includes the entire cellview if set to nil.

Value Returned

l_objects	The set of objects that match the search criteria.
nil	Finding objects was unsuccessful.

Examples

Returns all the nand2 instances in the flipflop schematic.

```
cvId = dbOpenCellViewByType( "sample" "flipflop" "schematic" "" 'r nil )
instList = schGetMatchingObjects( cvId "cellName" "==" "nand2" nil )
```

schGetPinOrder

```
schGetPinOrder(
     d_cvId
     )
     => l_pinList
```

Description

Returns the pin list, as defined in the portOrder property (if present) or as the default pin list.

Arguments

d_cvId	The cellview whose pin order you want retrieved.

Return Value

l_pinList	The pin list of the cellview, as defined in the portOrder property (if present) or as the default pin list.

Examples

Gets the pin order list for `cv`.

```
cv = geGetEditCellView( )
pinList = schGetPinOrder(cv)
```

schGetPostCheckTriggers

```
schGetPostCheckTriggers(
     )
     => list / nil
```

Description

Lists all the post-check triggers registered using schRegPostCheckTrigger.

Arguments

None

Return Value

List	List of post-check triggers registered using schRegPostCheckTrigger.
nil	No registered post-check triggers.

Examples

If there are two registered post-check triggers:

```
_cphPostSchExtractTrigger
_schCICheck
```

Then the call `schGetPostCheckTriggers` will return:

```
("_cphPostSchExtractTrigger"
"_schCICheck")
```

Related Topics

schRegPostCheckTrigger

schGetPreCheckTriggers

```
schGetPreCheckTriggers(
     )
     => list / nil
```

Description

Lists all the pre-check triggers registered using `schRegPreCheckTrigger`.

Arguments

None

Return Value

List	List of post-check triggers registered using schRegPreCheckTrigger.
`nil`	No registered pre-check triggers.

Examples

If there are two registered pre-check triggers:

```
_cphPreSchExtractTrigger
_schCICheck
```

Then the call `schGetPreCheckTriggers` will return:

```
("_cphPreSchExtractTrigger"
"_schCICheck")
```

Related Topics

schRegPreCheckTrigger

schGetPropertyDisplay

```
schGetPropertyDisplay(
     ?object d_object
     [?name S_name]
     )
     => l_textDisplays / s_visibility / nil
```

Description

Returns the attribute, property, and parameter textDisplays enabled for an object.

Arguments

`?object` *`d_object`*	The object whose display characteristics you want to view.
`[?name` *`S_name`*`]`	Name of the element of *`d_object`* whose display characteristics you want to view. If this argument is not specified, the function returns all available textDisplays of *`d_object`*.

Return Value

`l_textDisplays`	A list of the requested textDisplays.
`s_visibility`	If `d_object` is a textDisplay, `s_visibility` is a symbol representing the visibility of the textDisplay.
`nil`	If the requested textDisplays could not be found.

Examples

Converts all `'name` textDisplays to `'both`

```
foreach(td schGetPropertyDisplay(?object inst)
when(schGetPropertyDisplay(?object td) == 'name
schSetPropertyDisplay(?object td 'both)
)
)
```

Repositions a displayed property.

```
when(tds = schGetPropertyDisplay(?object inst ?name "libName")
td~>xy = 1.0:2.0
```

```
)
```

Related Topics

schSetPropertyDisplay

schGetShapeStyle

```
schGetShapeStyle(
     d_shape
     )
     => l_style
```

Description

Returns a Disembodied property list (DPL) containing the style attributes of a given shape.

Arguments

d_shape	Database shape Id.

Value Returned

l_style	A DPL containing each of the style attributes – `color`, `lineStyle`, `stipple`, `fillStyle`, `fillColor`

Examples

Selects a shape on the canvas and passes it as an argument to `schGetShapeStyle`:

```
schGetShapeStyle(car(selectedSet()))
=> (nil color "cadetBlue" lineStyle "solid" stipple "solid" fillStyle "solid"
fillColor "cadetBlue")
```

schGetSignalTypeIntegrity

```
schGetSignalTypeIntegrity(
     t_netSigType
     t_termNetSigType
     t_termDirection
     )
     => t_severity / nil
```

Description

Gets the conflict severity for signal type integrity check for connections between nets of specified signal type, and instance pins of specified signal type and direction.

Arguments

t_netSigType	The specified signal type.
t_termNetSigType	The terminal net signal type.
t_termDirection	The terminal direction.

Value Returned

t_severity	Conflict severity.
nil	Command failed.

Examples

```
schGetSignalTypeIntegrity("tieOff" "scan" "inputOutput")
==> "error"
```

Related Topics

schSetSignalTypeIntegrity

schGetSplitInstances

```
schGetSplitInstances(
     d_instId
     )
     => l_splitInstIds / nil
```

Description

Returns a list of split instance IDs that are associated with the given split-primary instance.

Arguments

d_instId	Split primary instance ID.

Value Returned

l_splitInstIds	Returns a list of split instances.
nil	Returning the list was unsuccessful.

Examples

Returns a list containing i0_s1, i0_s2, i0_s3.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "r")
s2 = dbOpenCellViewByType("lib" "bga" "s2" "schematicSymbol" "r")
s3 = dbOpenCellViewByType("lib" "bga" "s3" "schematicSymbol" "r")
i0_s1 = schCreateInst(cvId s1 "I0" 0:1 "R0")
i0 = schGetSplitPrimaryInst(i0_s1)
i0_s2 = schCreateInst(cvId s2 "I0" 0:2 "R0")
i0_s3 = schCreateInst(cvId s3 "I0" 0:3 "R0")
schGetSplitInstances(i0)
```

schGetSplitInstTerms

```
schGetSplitInstTerms(
     d_instTermId
     )
     => l_splitInstTermIds / nil
```

Description

Returns a list of split instance terminals that have the same name as that of the given split-primary instance.

Arguments

d_instTermId	Split-primary instance terminal ID.

Value Returned

l_splitInstTermIds	Returns a list of split instance terminals.
nil	Returning the list was unsuccessful.

Examples

Returns a list of split instance terminals with the same name as the primary instance term name.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "r")
i0_s1 = schCreateInst(cvId s1 "I0" 0:1 "R0")
i0 = schGetSplitPrimaryInst(i0_s1)
primTerm = car(i0->instTerms)
schGetSplitInstTerms(primTerm)
```

schGetSplitPrimaryInst

```
schGetSplitPrimaryInst(
     d_splitInstId
     )
     => d_inst / nil
```

Description

Returns the split-primary instance associated with the given split symbol.

Arguments

d_splitInstId	The ID of the split instance.

Value Returned

d_inst	Returns the split-primary instance.
nil	Returning the instance was unsuccessful.

Examples

Returns the split-primary, i0.

```
schGetSplitPrimaryInst(i0_s1)
```

schGetSplitPrimaryInstTerm

```
schGetSplitPrimaryInstTerm(
     d_splitInstTermId
     )
     => d_instTerm / nil
```

Description

Returns the terminal of the split-primary instance that has the same terminal name as the specified split instance terminal.

Arguments

d_splitInstTermId	Split instance terminal ID.

Value Returned

d_instTerm	Returns the terminal of the split-primary instance.
nil	Returning the terminal was unsuccessful.

Examples

Returns the corresponding terminal from split-primary.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "r")
i0_s1 = schCreateInst(cvId s1 "I0" 0:1 "R0")
splitTerm = car(i0_s1->instTerms)
schGetSplitPrimaryInstTerm(splitTerm)
```

schGetWireColor

```
schGetWireColor(
     d_wireId
     )
     => t_colorName / nil
```

Description

Returns the color used to draw a wire segment.

Arguments

d_wireId	The wire segment ID.

Value Returned

t_colorName	The wire's color. Colors are referred to by the names defined in the Display Resource File.
nil	Either the wireId is not a wire segment or the color of the wire could not be retrieved.

Examples

```
schGetWireColor( car( geGetSelectedSet () ) )
```

Related Topics

schSetWireColor

schGetWireLineStyle

schSetWireLineStyle

schCreateWire

schGetWireLineStyle

```
schGetWireLineStyle(
     d_wireId
     )
     => t_lineStyleName / nil
```

Description

Returns the line style used to draw a wire segment.

Arguments

d_wireId	The wire segment ID.

Value Returned

t_lineStyleName	The wire's line style. Line styles are referred to by the names defined in the Display Resource File.
nil	Either the wireId is not a wire segment or the line style of the wire could not be retrieved.

Examples

```
schGetWireLineStyle( car( geGetSelectedSet () ) )
```

Related Topics

schSetWireLineStyle

schGetWireColor

schSetWireColor

schCreateWire

schGlueLabel

```
schGlueLabel(
     d_label
     d_figure
     )
     => t / nil
```

Description

Glues the label to the figure you specify using a database child/parent relationship in which the label is the child. You can glue a pin label to a pin only when the label defines the name for that pin. You can glue a wire label only to a wire, a pin of a component, or a pin of a schematic. The label defines the name of the net associated with the wire, the pin of the component, or the pin of the schematic. You can glue note labels to any object.

Arguments

d_label	ID of the label to glue.
d_figure	ID of the figure on which to glue the label.

Value Returned

`t`	Glued the label to the figure you specify using a database child/parent relationship in which the label is the child.
`nil`	Gluing the label was unsuccessful.

Examples

Glues the label designated by `labelId` to the figure designated by `figId`.

```
schGlueLabel( labelId figId )
```

schHdlPrintFile

```
schHdlPrintFile(
     )
     => t
```

Description

Prints the current HDL file.

Arguments

None

Value Returned

Always returns `t`.

schHdlPrintVars

```
schHdlPrintVars(
     )
     => t
```

Description

Prints the current values of the schematic HDL variables.

Arguments

None

Value Returned

Always returns `t`.

Related Topics

Customizing the Virtuoso Schematic Editor

schHDLReturn

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

Description

Returns up the hierarchy from a Verilog view window.

Usable when viewing `verilog` after completing a descend action from a schematic.

Arguments

w_windowId	Window where the function runs. If not specified, the current window is used.

Value Returned

Always returns `t`.

Examples

Displays the parent view of the cellview in the specified window. The parent view is displayed in the current window or an existing window depending on whether you have turned on the *Create New Window When Descending* option on the User Preferences form.

```
schHDLReturn( )
```

schIgnore

```
schIgnore(
     ?objectId d_instId
     ?setIgnore g_setIgnore
     )
     => t / nil
```

Description

Adds or removes the ignore properties from the specified instance. You can view the registered ignore properties on the *Ignore Properties* tab by clicking *Options – Editor.*

Arguments

`?objectId` *`d_instId`*

Instance ID for which ignore properties are to be added or removed.

`?setIgnore` *`g_setIgnore`*

`t`: Adds the ignore properties on the instance.

`nil`: Removes the ignore properties from instance.

Value Returned

`t`	Addition or removal of ignore properties was successful.
`nil`	Operation was unsuccessful

Examples

The following example sets the ignore properties on the instance currently selected in the canvas.

```
schIgnore(?objectId car(geGetSelSet()) ?setIgnore t)
```

The following example sets the ignore properties on the instance specified by `instId`.

```
schIgnore(?objectId instId ?setIgnore t)
```

The following example removes the ignore properties from the instance currently selected in the canvas.

```
schIgnore(?objectId car(geGetSelSet()) ?setIgnore nil)
```

Related Topics

Ignoring Instances

schInhConFind

```
schInhConFind(
     w_windowId
     [ d_inst ]
     )
     => l_inhConList
```

Description

Given a `windowId` and an optional list of instances (or all instances in the window if none are explicitly specified), will return a list of inherited connections eligible for override beneath these instances.

Each inherited connection in the list is represented by a DPL (disembodied property list) with the following fields:

- `name`

  The name of the inherited connection (the “property name” that must be used to override the connection).

- `default`

  The default net name.

- `value`

  The current net to which the connection is attached.

- `inst`

  The instance under which the connection was found.

Arguments

w_windowID	Window where the function runs. If not specified, the current window is used.
[d_inst]	Optional list of instances whose eligible inheritance connection details you want to return.

Value Returned

l_inhConList	DPL list of each returned inherited connection.

Examples

```
schInhConFind(hiGetCurrentWindow())
=>
((nil name POWER default pwr! value pwr! inst db:12345678)
(nil name GROUND default gnd! value gnd! inst db:12345678))
```

schInhConSet

```
schInhConSet(
     t_inhCon
     [ ?name t_name ]
     [ ?default t_default ]
     [ ?value t_value ]
     [ ?pinName t_pinName ]
     [ ?pinDir t_pinDir ]
     [ ?pinPos t_pinPos ]
     )
     => t / nil
```

Description

Manipulates inherited connections located using schInhConFind.

You can use this function to:

- override a connection by passing the name of a net to connect to.
- convert a connection to a local pin with no associated net expression.
- convert to an inherited pin (with potentially different connection parameters).
- change the connection name or default net.

Arguments

t_inhCon	The given inherited connection.
?name *t_name*	If not nil, re-parameterizes the connection using this new name. If nil, the connection retains its original name.
?default *t_default*	Re-parameterizes the connection with the specified default net name, or leaves it as is if nil.
?value *t_value*	Specifies the name of a local or global net to connect the connection to.
?pinName *t_pinName*	Specifies the name of a pin that is to be used to connect to the inherited connection. The pin is automatically created.
?pinDir *t_pinDir*	If pinName has been specified will give the required direction of the pin.
?pinPos *t_pinPos*	If pinName has been specified, this is the x:y location of the created pin.

Value Returned

`t`	Inheried connection successfully modified.
`nil`	Modification was unsuccessful.

Examples

Note: These examples assume that `inhCon` is an entry in a list previously returned by schInhConFind. If we therefore have:

```
inhCon = '(nil name "POWER" default "pwr!" value "pwr" inst db:12345678)

; Override a connection by connecting to a local net.
; Creates netSet on inhCon->inst:
; name=inhCon->name, value=myPower.
schInhConSet(inhCon ?value myPower)

; Override by connecting to a global net.
; Creates netSet on inhCon->inst:
; name=inhCon->name, value=vdd!
schInhConSet(inhCon ?value vdd!)

; Convert to a schematic pin.
; Creates pin VDD in inhCon->insts cellview, and adds netSet to inst:
; name=inhCon->name, value=VDD
schInhConSet(inhCon ?pinName VDD ?pinPos 0:0)

; Change the name of a connection, but leave default net alone.
; Creates netSet on inhCon->inst:
; name=inhCon->name, value=[@VDD:%:pwr!]
schInhConSet(inhCon ?name VDD)

; Change default net, leave name alone.
; Creates netSet on inhCon->inst:
; name=inhCon->name, value=[@POWER:%:vdd!]
schInhConSet(inhCon ?default vdd!)

; Change both name and default value.
; Creates netSet on inhCon->inst:
; name=inhCon->name, value=[@VDD:%:vdd!]
schInhConSet(inhCon ?name VDD ?default vdd!)

; Propagate inherited connection via inherited pin.
; Creates pin VDD in inhCon->insts cellview, and adds netSet to inst:
schInhConSet(inhCon ?pinName VDD ?pinPos 0:0 ?name VDD ?default vdd!)

; Convert all inherited connections to pins.
procedure(pinPos() /* Generate a pin position. */ )
foreach(inhCon schInhConFind(hiGetCurrentWindow())
schInhConSet(inhCon ?pinName inhCon->name ?pinPos pinPos())

)

; ...etc...
```

Related Topics

schInhConFind

schInstallHDL

```
schInstallHDL(
     g_library
     t_cellName
     t_viewName
     t_srcName
     [ g_createSymbol ]
     [ g_overrideIfExist ]
     )
     => t / nil
```

Description

Installs a Verilog HDL source file as an HDL cellview and creates the cell, view, and cellview objects in the library if necessary. This function can also create a matching symbol cellview.

Arguments

g_library	Either a library name string or a library identifier returned by `ddGetObj`.
t_cellName	Name of the cell.
t_viewName	Name of the view.
t_srcName	Path to the Verilog HDL source file; must be enclosed in quotation marks.
g_createSymbol	Boolean flag that specifies whether a matching symbol is created.
g_overrideIfExist	Boolean flag that specifies whether the specified cellview can be replaced if it already exists. The default value is `nil`. When the default value is specified, an error message is displayed if the specified cellview already exists. When the value is `t`, the cellview gets replaced.

Value Returned

`t`	Installed a Verilog HDL source file as an HDL cellview and created the cell, view, and cellview objects in the library if necessary.
`nil`	Verilog HDL source file installation was unsuccessful.

Examples

Creates the HDL cellview `myDesign functional` in the library `myLib`.

```
lib = ddGetObj( "myLib" )
schInstallHDL( lib "myDesign" "functional" "myDesign.v" )
```

In addition to creating the HDL cellview `myDesign functional` in the library `myLib`, it creates a matching `myDesign` symbol cellview and overwrites the symbol if it already exists.

```
schInstallHDL( lib "myDesign" "functional" "myDesign.v" t t)
```

schInstToView

```
schInstToView(
     d_inst
     t_viewTo
     t_fromFunc
     t_toFunc
     )
     => t / nil
```

Description

Generates a cellview type from an instance of a symbol.

The instance master and the destination view must have the same library and cell name.

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for the schViewMasters list of translation functions and documentation for creating your own translation functions.

Arguments

d_inst	Instance ID from a schematic to use as the source for the translation.
t_viewTo	Name of the destination view; must be enclosed in quotation marks.
t_fromFunc	Name of the SKILL procedure to translate from the instance master to the pin list intermediate format; must be enclosed in quotation marks.
t_toFunc	Name of the SKILL procedure to translate from the pin list intermediate format to the destination view; must be enclosed in quotation marks.

Value Returned

`t`	Generated a cellview type from an instance of a symbol.
`nil`	Cellview type generation was unsuccessful.

Examples

Generates a Verilog HDL shell from the specified instance.

```
schInstToView( inst "functional" "schSymbolToPinList" "schPinListToVerilog" )
```

schIsFlightLine

```
schIsFlightLine(
     d_flightId
     )
     => t / nil
```

Description

Checks if the specified object is a flight line on a net.

Arguments

`d_flightId`	The object to be confirmed as a flight line on a net.

Value Returned

`t`	Confirmed that the object is a flight line on the net.
`nil`	Check was unsuccessful.

Examples

Click a flight line in a schematic window, followed by these commands:

```
obj = css()
schIsFlightLine(obj)
```

```
> t
```

Click an instance in a schematic window, followed by the same commands:

```
obj = css()
schIsFlightLine(obj)
```

```
> nil
```

schIsHDLCapEnabled

```
schIsHDLCapEnabled(
     { t | nil }
     )
     => t / nil
```

Description

Validates that a license supporting the schematic editing feature is already checked out and does a recheck to ensure that the license has not timed out. Otherwise, attempts to check out a license.

Arguments

`t`	Specifies that the application-specific error message or the original License Manager error message should be issued.
`nil`	Specifies that no message should be issued.

Value Returned

`t`	Validated that a license supporting the schematic editing feature is already checked out and does a recheck to ensure that the license has not timed out.
`nil`	Validation was unsuccessful.

Examples

Validates that a license supporting the schematic editing feature is already checked out.

```
schIsHDLCapEnabled( t )
```

schIsInCheckHier

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

Description

Identifies and performs the specific actions, as mentioned in the SKILL code, when checking a design hierarchy as opposed to a single cellview during a schematic hierarchy check. When you check a design hierarchy or a single cellview, the schematic checker runs any pre or post check triggers and custom checker rules that you have previously registered.

Arguments

None

Value Returned

`t`	The hierarchy checker is currently running.
`nil`	The command is unsuccessful or an error occurred.

Examples

Register a trigger function that runs before schematic checking begins.

```
schRegPreCheckTrigger('myPreCheckTrigger)

procedure(myPreCheckTrigger(cv)
    if(schIsInCheckHier()
    then
        info("Pre-check hierarchy.\n")
    else
        info("Pre-check cellview.\n")
    )
)
```

Gets the cellview to check.

```
cv = geGetEditCellView()
```

Checks the single cellview.

```
schCheck(cv) ;; prints "Pre-check cellview."
```

Checks the hierarchy starting from cellview.

```
schCheckHier(cv "schematic" "analogLib basic") ;; prints "Pre-check hierarchy."
```

Checks the configured hierarchy.

```
schCheckHierConfig(deGetConfigId()) ;; prints "Pre-check hierarchy."
```

Related Topics

schCheck

schCheckHier

schCheckHierConfig

schRegPostCheckTrigger

schRegPreCheckTrigger

schRegisterCheckRule

schIsIndexCV

```
schIsIndexCV(
     d_cvId
     )
     => t / nil
```

Description

Tests whether the given cellview is an index schematic cellview.

Arguments

d_cvId	Cellview ID of the cellview to be tested.

Value Returned

t	Tested whether the given cellview is an index schematic cellview.
nil	Unsuccessful.

schIsSchEditOk

```
schIsSchEditOk(
     d_cvId
     [ ?skipWritableCheck g_skipWritableCheck ]
     [ ?dialog g_dialog ]
     )
     => t / nil
```

Description

Checks if the given cellview is a schematic view, whether it is writable, and also whether the edit capability (VSE license successfully checked out) is enabled.

Arguments

d_cvId	Cellview ID of the cellview to be tested.
?skipWritableCheck *g_skipWritableCheck*	Does not test whether *d_cvId* is writable.
?dialog *[g_dialog]*	Specifies whether a warning message is displayed. ■ When set to `nil`, messages are not presented (unless the check results are nil). ■ When set to `t`, messages are presented in a dialog box.

Value Returned

`t`	Tested whether the given schematic cellview is writable and whether the edit capability is enabled.
`nil`	Check was unsuccessful.

schIsSheetCV

```
schIsSheetCV(
     d_cvId
     )
     => t / nil
```

Description

Tests whether the given cellview is a multisheet schematic cellview.

Arguments

d_cvId	Cellview ID of the cellview to be tested.

Value Returned

t	Tested whether the given cellview is a multisheet schematic cellview.
nil	Test was unsuccessful.

schIsSplitInst

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

Description

Tests if a specified instance is a split instance. An instance is a split instance if its master is a split symbol.

Arguments

d_instId	The schematic instance ID.

Value Returned

t	Confirmed that the instance is a split instance.
nil	Test was unsuccessful.

Examples

`i0_s1` is an instance in a schematic. The following example checks if `i0_s1` is a split instance.

```
schIsSplitSymbol(i0_s1->master)
=> t
schIsSplitInst(i0_s1)
=> t
```

schIsSplitPrimaryInst

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

Description

Tests if a specified instance is a split-primary instance.

Arguments

d_instId	The schematic instance ID.

Value Returned

`t`	Confirmed that the instance is a split-primary instance.
`nil`	Test was unsuccessful.

Examples

Checks if `i0` is a split-primary instance.

```
schIsSplitPrimarySymbol(i0->master)
=> t
schIsSplitPrimaryInst(i0)
=> t
```

schIsSplitPrimarySymbol

```
schIsSplitPrimarySymbol(
     d_cvId
     )
     => t / nil
```

Description

Tests if a specified cellview is a split-primary symbol.

Arguments

d_cvId	The cellview ID of a symbol view.

Value Returned

`t`	The cellview is a split-primary symbol.
`nil`	Test was unsuccessful.

Examples

Checks if the cellview `lib bga symbol` is a split-primary symbol.

```
sym = dbOpenCellViewByType("lib" "bga" "symbol" "schematicSymbol" "r")
schIsSplitPrimarySymbol(sym)
=> t
```

schIsSplitSymbol

```
schIsSplitSymbol(
     d_cvId
     )
     => t / nil
```

Description

Tests if a specified cellview is a split symbol.

Arguments

d_cvId	The cellview ID of a symbol view.

Value Returned

`t`	The cellview is a split symbol.
`nil`	Test was unsuccessful.

Examples

Checks if the cellview `lib bga s1` is a split symbol.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "r")
schIsSplitSymbol(s1)
=> t
```

schIsSymEditOk

```
schIsSymEditOk(
     d_cvId
     [ ?skipWritableCheck g_skipWritableCheck ]
     [ ?dialog g_dialog ]
     )
     => t / nil
```

Description

Checks if the given cellview is a schematic symbol view, whether it is writable, and also whether the edit capability (VSE license successfully checked out) is enabled.

Arguments

d_cvId	Cellview ID of the cellview to be tested.
?skipWritableCheck *g_skipWritableCheck*	Does not test whether *d_cvId* is writable.
[g_dialog]	Optional argument that specifies whether a warning message is displayed. ■ When set to `nil`, messages are not presented (unless the check results are nil). ■ When set to `t`, messages are presented in a dialog box.

Value Returned

`t`	Tested whether the given schematic symbol cellview is writable and whether the edit capability is enabled.
`nil`	Check was unsuccessful.

schIsTextEditable

```
schIsTextEditable(
     d_databaseID
     )
     => t / nil
```

Description

Queries whether or not a label (optionally inside an instance) can be edited directly on the canvas.

Arguments

d_databaseID	The database ID of a label (and optionally, the owning instance ID).

Value Returned

`t`	Label is editable.
`nil`	Label is not editable.

schIsUsingSplitFeature

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

Description

Indicates if a specified cellview uses the split feature. This implies that a schematic cellview has split instances with correctly set split-primary instances and a symbol cellivew is a split or has a split-primary symbol.

Arguments

d_cellViewId	The ID of an editable cellview with `viewType` as `schematicSymbol`.

Value Returned

`t`	If the specified schematic cellview has at least one split instance or the specified `schematicSymbol` cellview is a split or split-primary symbol.
`nil`	Check was unsuccessful.

Examples

Checks if the specified `schematicSymbol` cellview is a split or split-primary symbol.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "r")
schIsUsingSplitFeature(s1)
schematic = dbOpenCellViewByType("lib" "test" "schematic" "schematic" "r")
schIsUsingSplitFeature(schematic)
=> t
```

schIsWire

```
schIsWire(
     d_wireId
     )
     => t / nil
```

Description

Checks if the specified object is a wire.

Arguments

d_wireId	The object to be confirmed as a wire on the net

Value Returned

t	Confirmed that the given object is a wire on a net.
nil	Check was unsuccessful.

Examples

Example: Click a wire segment in a schematic window, followed by these commands:

```
obj = css()
schIsWire(obj)
```

```
> t
```

Click an instance in a schematic window, followed by the same commands:

```
obj = css()
schIsWire(obj)
```

```
> nil
```

schIsWireLabel

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

Description

Tests whether the given database figure is a schematic wire label.

Arguments

d_figId	The database ID of a figure.

Value Returned

t	Tested whether the given database figure is a schematic wire label.
nil	Test was unsuccessful.

schLayoutToPinList

```
schLayoutToPinList(
     t_libName
     t_cellName
     t_viewName
     )
     => g_pinList
```

Description

Translates a `layout` cellview into an intermediate pin list format. The pin list represents all of the terminals in the layout and their directions. The pin list also represents the cellview level properties in the maskLayout.

Arguments

t_libName	Library containing the `maskLayout` cellview to translate; must be enclosed in quotation marks.
t_cellName	Cell containing the `maskLayout` cellview to translate; must be enclosed in quotation marks.
t_viewName	View containing the `maskLayout` cellview to translate; must be enclosed in quotation marks.

Value Returned

g_pinList	Terminal and property information organized in a pin list.

Examples

```
pinList = schLayoutToPinList( myLib myDesign layout )
```

where

```
pinList = `( nil ports (( nil name "a" direction "input" )
    ( nil name "b" direction "input" )
    ( nil name "c" direction "output" )
    )
)
```

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '( nil ports portList
     [prop proplist] )
portlist = ( termDef termDef...termDef )
termDef = (nil name termName
     direction termDir
     [prop propList]
     [pins termPins]
     )
proplist = ( nil propName propValue
     propName propValue
     ...
     )
termPins = ( pinDef pinDef...pinDef )
pinDef =  ( nil name pinName [accessDir accessDir] )
```

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for usage of schLayoutToPinList in the schViewMasters list of translation functions.

schMapGetOption

```
schMapGetOption(
     s_arg
     )
     => t_value / nil
```

Description

Returns the value that is internally registered to the provided argument of the Schematic Migration configuration file.

Arguments

s_arg	Symbolic name of the argument.

Value Returned

t_value	Internal value of the provided argument.
nil	Internal value could not be given.

Examples

```
schMapGetOption('runDir)
```

schMapLoadConfig

```
schMapLoadConfig(
     t_path
     )
     => t / nil
```

Description

Loads the configuration file for Schematic Migration.

Arguments

t_path	Path and filename of the configuration file.

Value Returned

t	Configuration file is loaded.
nil	Configuration file could not be loaded.

Examples

```
schMapLoadConfig("./SCHMAP_CONFIG/schMapDefaults.il")
```

schMapMCCleanup

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

Description

Cleans up the migration running directory from the directories and files generated during migration. Additionally, it removes definitions and folders of temporary libraries.

This SKILL function is equivalent to the *Multiple Cells Mapping Cleanup* option. In CIW, this option is available in *Tools – Migrate – Multiple Schematics Mapping.*

Arguments

None

Value Returned

`t`	Cleanup process is complete.
`nil`	Cleanup process is not complete.

Examples

```
schMapMCCleanup()
=> t
```

Related Topics

schMapMCCreateTargets

schMapMCFinalizeTargets

schMapMCSaveSourceData

schMapMCSettings

schMapMCCreateTargets

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

Description

Creates targets for routing after placing the missing devices in the target cellviews.

This SKILL function uses results from `schMapMCSaveSourceData` and is equivalent to the *Create Multiple Target Schematics (Part 2)*. In CIW, this option is available in *Tools – Migrate – Multiple Schematics Mapping.*

Arguments

None

Value Returned

`t`	All targets are created succesfully.
`nil`	Targets are not created.

Examples

```
schMapMCCreateTargets()
=> t
```

Related Topics

schMapMCCleanup

schMapMCFinalizeTargets

schMapMCSaveSourceData

schMapMCSettings

schMapMCFinalizeTargets

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

Description

Finalizes the assembly of the target cells and libraries. This SKILL function is equivalent to the *Finalize Multiple Target Schematics (Part 3)* option. In CIW, this option is available in *Tools – Migrate – Multiple Schematics Mapping.*

Arguments

None

Value Returned

`t`	All target cells and libraries are finalized succesfully.
`nil`	Targets are not finalized.

Examples

```
schMapMCFinalizeTargets()
=> t
```

Related Topics

schMapMCCleanup

schMapMCCreateTargets

schMapMCSaveSourceData

schMapMCSettings

schMapMCSaveSourceData

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

Description

Prepares all the cells, which are defined in the cell list file for migration. This SKILL function is equivalent to the *Save Multiple Source Schematics Data (Part 1)* option. In CIW, this option is available in *Tools – Migrate – Multiple Schematics Mapping.*

This SKILL function also indicates that the schematic templates from the original cells that contain only wires and unmodified instances are ready to be processed by `schMapMCCreateTargets()`.

Arguments

None

Value Returned

`t`	All source data are prepared for migration.
`nil`	All source data is not prepared for migration.

Examples

```
schMapMCSaveSourceData()
=> t
```

Related Topics

schMapMCCleanup

schMapMCFinalizeTargets

schMapMCCreateTargets

schMapMCSettings

schMapMCSettings

```
schMapMCSettings (
     [?rundir x_rundir]
     [?mcrundir x_inputdatadir]
     [?tgtlibsdir x_targetlibsdir]
     [?cellsfile x_cellfilename]
     [?tgtprefix x_temptargetlibsprefix]
     [?dfntgtlibsbn g_redefinetargetlibs]
     )
     => t
```

Description

Assigns values to the settings that are used to carry out the batch multicell schematic migration. This SKILL function is equivalent to the *Multiple Schematics Mapping – Settings* option. In CIW, this option is available in *Tools – Migrate.*

Arguments

`?rundir` *`x_rundir`*

Name of the working directory.

`?mcrundir` *`x_inputdatadir`*

Name of the input data directory.

`?tgtlibsdir` *`x_targetlibsdir`*

Name of the target library directory.

`?cellsfile` *`x_cellfilename`*

Name of the cell file.

`?tgtprefix` *`x_temptargetlibsprefix`*

Name of the temporary target libaries prefix.

`?dfntgtlibsbn` *`g_redefinetargetlibs`*

Specifies whether to redefine the target libraries.

Value Returned

`t`	Values are assigned.

Examples

```
schMapMCSettings(
?rundir "myRunDir"
?mcrundir "myMcRunDir"
?tgtlibsdir "myTgtLibsDir"
?cellsfile "myCellsFile"
?tgtprefix "myPrefix"
?dfntgtlibsbn "yes"
)
=> t
```

Related Topics

schMapMCCleanup

schMapMCFinalizeTargets

schMapMCCreateTargets

schMapMCSaveSourceData

schMapSaveConfig

```
schMapSaveConfig(
     t_path
     )
     => t / nil
```

Description

Creates a configuration file with the arguments used in Schematic Migration that have values that differ from the default.

Arguments

t_path	Path and filename of the configuration file to be saved.

Value Returned

t	Configuration file is created successfully.
nil	Configuration file could not be created.

Examples

```
schMapSaveConfig("./schMapDefaults.il")
```

schMapSetOption

```
schMapSetOption(
     s_arg
     x_value
     )
     => t_value / nil
```

Description

Assigns values to the arguments of the Schematic Migration configuration file.

Arguments

s_arg	Symbolic name of the argument.
x_value	New value of argument.

Value Returned

t_value	Newly assigned value of the provided argument.
nil	Newly assigned value could not be given.

Examples

```
schMapSetOption('runDir "./SCHMAP_RUNDIR")
```

schMouseApplyOrFinish

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

Description

Adds a point and applies or finishes the active enter function command based on the setting of the `modalCommands` schematic environment variable. It is designed to be used for double-clicking with the left mouse button.

Arguments

None

Value Returned

`t`	A point is added and the active enter function command is applied or finished based on the setting of the `modalCommands` schematic environment variable.
`nil`	Adding a point was unsuccessful.

Examples

Binds the left mouse button double-click action during schematic editor enter function commands to `schMouseApplyOrFinish`.

```
hiSetBindKey( "Schematics" "None<Btn1Down>(2) EF" "schMouseApplyOrFinish()" )
```

schMove

```
schMove(
     d_fig
     d_destCV
     l_transform
     )
     => d_object / nil
```

Description

Moves the object you specify to a destination cellview. The object location and orientation can be specified before the object is placed at the destination location by the given transformation argument. The copied figure is first rotated and reflected about the origin as specified by the orientation of the transform, then translated by the offset of the transform.

The destination cellview must be editable. This function moves figures between schematic or symbol cellviews only.

Arguments

d_fig	Figure to move.
d_destCV	Destination schematic or symbol cellview in which to place the object.
l_transform	Specifies the relative location, orientation, and optionally magnification of the moved figure, specified as a list of the form: (*l_offset t_orient [n_magnification]*) Where: *l_offset* is the offset from the original position expressed as a list of two floats, the first specifying the distance to move in the x direction and the second the distance in the y direction; for example (10.0:5.0). *t_orient* specifies the orientation of the moved object and is one of R0, R90, R180, R270, MX, MXR90, MY, MYR90. The value must be enclosed in double quotes. *n_magnification* specifies the relative size of the moved object. The default is 1.0 (i.e. the same size as before the move).

Value Returned

d_object	The ID of the figure after it is moved.
nil	Could not move the object.

Examples

Moves `fig1` to the cellview `cv2`; the offset for `fig1` is 10.0,5.0 and `fig1` is rotated 90 degrees from the original orientation. The resulting `figId` is returned and assigned to `fig1`.

```
fig1 = schMove( fig1 cv2 list(10.0:5.0 "R90" ) )
```

schNetExprAvailProps

```
schNetExprAvailProps(
     l_designSpec
     l_instPaths
     )
     => l_availProps
```

Description

Returns a list of available properties, and their evaluated values, for the various occurrences passed.

Arguments

l_designSpec — A DPL with the format:

```
'(nil
libname t_libname
cellName t_cellname
viewName t_viewName
switchViewList t_switchViewList
stopViewList t_stopViewList)
```

Note: If the `viewName` is a configuration cellview, then `switchViewList` and `stopViewList` are optional, and will be ignored if set. The design configuration is based on the configuration file.

l_instPaths — A list of full instance names in the hierarchy for which data is requested.

Note: Wherever possible you should pass a list of instances to these functions, rather than call functions multiple times to get results for specific `instPaths`, as each cell will lead to a design traversal and can be time consuming.

An example of an `instPaths` argument value is:

```
'("/I0" "/I1/MN0")
```

Value Returned

l_availProps	Returns a list of available properties, and their evaluated values, for the various occurrences passed to the function.
	
	For example:
	`'(((“gnd” “gnd!”)`
	`(“vdd” “vdd!” ))`
	`((“gnd” “gnd!”)`
	`(“vdd” “vdd!”))`
	`)`

Examples

See *Example* section for `schNetExprEvalNames`.

Related Topics

schNetExprEvalNames

schNetExprEvalNames

```
schNetExprEvalNames(
     l_designSpec
     l_instPaths
     [ ?listCellView g_listCellView ]
     [ ?listOccurrences g_listOccurrences ]
     )
     => l_netExprEvalNames
```

Description

Returns a list of evaluated names for all occurrences specified.

Arguments

l_designSpec A DPL with the format:

```
'(nil
libname t_libname
cellName t_cellname
viewName t_viewName
switchViewList t_switchViewList
stopViewList t_stopViewList)
```

Note: If the `viewName` is a configuration cellview, then `switchViewList` and `stopViewList` are optional, and will be ignored if set. The design configuration is based on the configuration file.

l_instPaths A list of full instance names in the hierarchy for which data is requested.

Note: Wherever possible you should pass a list of instances to these functions, rather than call functions multiple times to get results for specific `instPaths`, as each cell will lead to a design traversal and may be time consuming.

An example of an `instPaths` argument value would be:

```
'("/I0" "/I1/MN0")
```

?listCellView *g_listCellView*

A boolean value to indicate whether cellview data is also required for each evaluated name.

?listOccurrences *g_listOccurrences*

A boolean value to indicate whether occurrence data is also required for each evaluated name.

Value Returned

l_netExprEvalNames Returns a list of evaluated names for all occurrences specified.

When occurrence or cellview data is not requested, the evaluated name data is an ordered tuple of: evaluated name, number of cellviews and the number of occurrences. For example:

```
...
("5V!" "1" "3")
...
```

When cellview data and/or occurrence data is requested for each evaluated name, the list will have five members as `cvInfo` followed by `occInfo` is appended. If only one data element is requested the corresponding entry for the omitted data is `nil`. Both `cvInfo` and `occInfo` are also lists:

- `cvInfo` is a list containing the `libName`, `cellName`, `viewName`, property name, default value, and the number of occurrences, for example:

  ```
  ...
  ("inhConnSmall" "pmos" "schematic" "gnd" "gnd!" 2)
  ...
  ```

- `occInfo` contains the `occPath` (another list), property name, and default value. The `occPath` is specified as a list of (`libName cellName viewName instName`). The last list entry will not have an `instName`, and refers to the switch instance master, for example:

  ```
  '(
      ( ( ("innConnSmall" "top" "schematic" "I4")
      ("inhConnSmall" "inv" "schematic" "P1")
      ("inhConnSmall" "pmos" "schematic")
  )
  "bulk_p"
  "vdd!"
  )
  ...)
  ```

Examples

```
ds = '( nil libName        "vanlib_cdb"
            cellName       "test_inh_pi"
            viewName       "schematic"
            switchViewList "schematic spectre"
            stopViewList   "spectre" )
schNetExprAvailProps( ds '("/M0" "/M1"))
==> returns:
(
    (("bulk_n" "gnd!"))
    (("bulk_n" "gnd!"))
)


schNetExprEvalNames( ds '("/M0" "/M1"))
==> returns:
 (
        (("gnd!" 1 1))
        (("gnd!" 1 1))
 )
schNetExprEvalNames( ds '("/M0" "/M1") ?listOccurrences t)
==> returns:
 (
        (("gnd!" 1 1 nil
            ((((("vanlib_cdb" "test_inh_pi" "schematic" "M0")
                        ("analogLib" "nmos" "spectre")
                    ) "bulk_n" "gnd!"
                )
            )
        )
        )
        (("gnd!" 1 1 nil
            ((((("vanlib_cdb" "test_inh_pi" "schematic" "M1")
                        ("analogLib" "nmos" "spectre")
                    ) "bulk_n" "gnd!"
                )
            )
        )
        )
 )
```

```
schNetExprEvalNames( ds '("/M0" "/M1") ?listCellView t)
==> returns:
 (
        (("gnd!" 1 1
            (("analogLib" "nmos" "spectre" "bulk_n" "gnd!"
                    1
                )
            ) nil
        )
        )
        (("gnd!" 1 1
            (("analogLib" "nmos" "spectre" "bulk_n" "gnd!"
                    1
                )
            ) nil
        )
        )
 )
schNetExprEvalNames( ds '("/M0" "/M1") ?listCellView t ?listOccurrences t)
==> returns:
 (
        (("gnd!" 1 1
            (("analogLib" "nmos" "spectre" "bulk_n" "gnd!"
                    1
                )
            )
            ((("vanlib_cdb" "test_inh_pi" "schematic" "M0")
                        ("analogLib" "nmos" "spectre")
                    ) "bulk_n" "gnd!"
                )
            )
        )
        )
        (("gnd!" 1 1
            (("analogLib" "nmos" "spectre" "bulk_n" "gnd!"
                    1
                )
            )
            ((("vanlib_cdb" "test_inh_pi" "schematic" "M1")
                        ("analogLib" "nmos" "spectre")
                    ) "bulk_n" "gnd!"
```

```
                    )
                )
            )
            )
)
```

schPinListToSchem

```
schPinListToSchem(
     t_libName
     t_cellName
     t_viewName
     g_pinList
     )
     => t / nil
```

Description

Generates a schematic cellview from a pin list.

Arguments

t_libName	Library containing the schematic to generate from the pin list; must be enclosed in quotation marks.
t_cellName	Cell containing the schematic to generate from the pin list; must be enclosed in quotation marks.
t_viewName	View containing the schematic to generate from the pin list; must be enclosed in quotation marks.
g_pinList	Terminal and property information to use in generating the target schematic.

Value Returned

t	Generated a schematic cellview from a pin list.
nil	Could not generate schematic cellview.

Examples

```
schPinListToSchem( "myLib" "myDesign" "schematic" pinList )
```

where

```
pinList = '(nil ports ((nil name "a" direction "input")
           (nil name "b" direction "input")
           (nil name "c" direction "output")
           )
)
```

Generates a schematic with two input pins, `"a"` and `"b"`, and one output pin, `"c"`.

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portList
    [prop proplist] )
portlist = (termDef termDef...termDef)
termDef = (nil name "termName"
    direction "termDir"
    [prop propList]
    [pins termPins]
    )
proplist = (nil propName propValue
    propName propValue
    ...
    )
termPins = (pinDef pinDef...pinDef)
pinDef =  (nil name "pinName"
    [accessDir "accessDir"])
```

schPinListToSchemGen

```
schPinListToSchemGen(
     t_libName
     t_cellName
     t_viewName
     g_pinList
     )
     => t / nil
```

Description

Opens the Create Schematic form if *t_viewName* is a schematic.

Arguments

t_libName	Library containing the schematic to generate from the pin list; must be enclosed in quotation marks.
t_cellName	Cell containing the schematic to generate from the pin list; must be enclosed in quotation marks.
t_viewName	View containing the schematic to generate from the pin list; must be enclosed in quotation marks.
g_pinList	Terminal and property information to use in generating the target schematic.

Value Returned

t	Generated a schematic cellview from a pin list.
nil	Could not open form.

Examples

```
schPinListToSchemGen( "myLib" "myDesign" "schematic" pinList )
```

where

```
pinList = '(nil ports ((nil name "a" direction "input")
    (nil name "b" direction "input")
    (nil name "c" direction "output")
    )
)
```

Generates a schematic with two input pins, "a" and "b", and one output pin, "c".

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portList
    [prop proplist] )
portlist = (termDef termDef...termDef)
termDef = (nil name "termName"
    direction "termDir"
    [prop propList]
    [pins termPins]
    )
proplist = (nil propName propValue
    propName propValue
    ...
    )
termPins = (pinDef pinDef...pinDef)
pinDef = (nil name "pinName"
    [accessDir "accessDir"])
```

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for usage of schPinListToSchemGen in the schPinListToViewReg list of translation functions.

schPinListToSymbol

```
schPinListToSymbol(
     t_libName
     t_cellName
     t_viewName
     g_pinList
     )
     => t / nil
```

Description

Generates a symbol cellview from a pin list.

Arguments

t_libName	Library containing the symbol to generate from the pin list; must be enclosed in quotation marks.
t_cellName	Cell containing the symbol to generate from the pin list; must be enclosed in quotation marks.
t_viewName	View containing the symbol to generate from the pin list; must be enclosed in quotation marks.
g_pinList	Terminal and property information to use in generating the target symbol.

Value Returned

t	Generated a symbol cellview from a pin list.
nil	Could not generate symbol cellview.

Examples

```
schPinListToSymbol( "myLib" "myDesign" "symbol" pinList )
```

where

```
pinList = '(nil ports ((nil name "a" direction "input")
    (nil name "b" direction "input")
    (nil name "c" direction "output")
    )
)
```

Generates a symbol with two input pins, `"a"` and `"b"`, and one output pin, `"c"`.

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portList
    [prop proplist] )
portlist = (termDef termDef...termDef)
termDef = (nil name "termName"
    direction "termDir"
    [prop propList]
    [pins termPins]
    )
proplist = (nil propName propValue
    propName propValue
    ...
    )
termPins = (pinDef pinDef...pinDef)
pinDef =  (nil name "pinName"
    [accessDir "accessDir"])
```

schPinListToSymbolGen

```
schPinListToSymbolGen(
     t_libName
     t_cellName
     t_viewName
     g_pinList
     )
     => t / nil
```

Description

Generates a symbol cellview from a pin list.

Arguments

t_libName	Library containing the symbol to generate from the pin list; must be enclosed in quotation marks.
t_cellName	Cell containing the symbol to generate from the pin list; must be enclosed in quotation marks.
t_viewName	View containing the symbol to generate from the pin list.; must be enclosed in quotation marks.
g_pinList	Terminal and property information to use in generating the target symbol.

Value Returned

t	Generated a symbol cellview from a pin list.
nil	Could not generate symbol cellview.

Examples

```
schPinListToSymbolGen( "myLib" "myDesign" "symbol" pinList )
```

where

```
pinList = `( nil ports ((nil name "a" direction "input" )
    ( nil name "b" direction "input" )
    ( nil name "c" direction "output" )
    )
)
```

Generates a symbol with two input pins, `"a"` and `"b"`, and one output pin, `"c"`.

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portList
    [prop proplist] )
portlist = (termDef termDef...termDef)
termDef = (nil name "termName"
          direction "termDir"
    [prop propList]
    [pins termPins]
    )
proplist = (nil propName propValue
    propName propValue
    ...
    )
termPins = (pinDef pinDef...pinDef)
pinDef =  (nil name "pinName"
    [accessDir "accessDir]")
```

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for usage of schPinListToSymbolGen in the schPinListToViewReg list of translation functions.

schPinListToVerilog

```
schPinListToVerilog(
     t_libName
     t_cellName
     t_viewName
     g_pinList
     )
     => t / nil
```

Description

Generates a Verilog HDL cellview from a pin list. The generated Verilog HDL cellview can be used with the Verilog integration.

Arguments

t_libName	Library containing the Verilog HDL cellview to generate from the pin list; must be enclosed in quotation marks.
t_cellName	Cell containing the Verilog HDL cellview to generate from the pin list; must be enclosed in quotation marks.
t_viewName	View containing the Verilog HDL cellview to generate from the pin list; must be enclosed in quotation marks.
g_pinList	Terminal and property information to use in generating the target Verilog HDL cellview.

Value Returned

t	Generated a Verilog HDL cellview from a pin list.
nil	Could not generate Verilog HDL cellview.

Examples

```
schPinListToVerilog( "myLib" "myDesign" "functional" pinList )
```

where

```
pinList = `( nil ports ((nil name "a" direction "input" )
    ( nil name "b" direction "input" )
    ( nil name "c" direction "output" )
    )
)
```

Generates the following Verilog module:

```
module myDesign ( "a", "b", "c" );
    input a;
    input b;
    output c;
endmodule
```

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portList
    [prop proplist] )
portlist = (termDef termDef...termDef)
termDef = (nil name "termName"
    direction "termDir"
    [prop propList]
    [pins termPins]
    )
proplist = (nil propName propValue
    propName propValue
    ...
    )
termPins = (pinDef pinDef...pinDef)
pinDef =  (nil name "pinName"
    [accessDir "accessDir"])
```

schPinListToView

```
schPinListToView(
     t_libName
     t_cellName
     t_viewName
     g_pinList
     t_toFunc
     )
     => t / nil
```

Description

Generates a cellview from a pin list.

Arguments

t_libName	Library containing the data to generate from the pin list; must be enclosed in quotation marks.
t_cellName	Cell containing the data to generate from the pin list; must be enclosed in quotation marks.
t_viewName	View containing the data to generate from the pin list; must be enclosed in quotation marks.
g_pinList	Terminal information to use in generating the target data.
t_toFunc	Name of the SKILL procedure to translate from the intermediate pin list format to the target; must be enclosed in quotation marks.

Value Returned

t	Generated a cellview from a pin list.
nil	Could not generate cellview.

Examples

```
schPinListToView( "myLib" "myDesign" "symbol" pinList "schPinListToSymbol" )
```

where

```
pinList = '( nil ports ((nil name "a" direction "input" )
                         nil name "b" direction "input" )
                       ( nil name "c" direction "output") )
```

Generates a symbol with two input pins, `a` and `b`, and one output pin, `c`.

The pin list format represents all the terminals and properties in the design and their directions; for example, `input` and `output`. The terminal and property information is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portlist
            [prop proplist] )
portlist =  (termDef termDef...termDef)
termDef =   (nil name "termName"
            direction termDir
            [prop proplist]
            [pins termPins]
            )
proplist =  (nil propName propValue
            propName propValue
            ...
            )
termPins =  (pinDef pinDef...pinDef)
pinDef =    (nil name "pinName"
                [accessDir "accessDir"])
```

See the *your_install_dir*`/tools/dfII/samples/local/schConfig.il` file for `schViewMasters` list of translation functions and documentation for creating your own translation functions.

schPlot

```
schPlot(
     [ t_file ]
     [ w_windowId ]
     )
     => t / nil
```

Description

Generates a plot. The plot is defined in the *t_file* plot template file. If you do not specify *t_file*, this function uses the plot options stored in the schPlotOptions property list. If you backannotate the schematic and you specify *w_windowId*, the generated plot has backannotated values.

A .cdsplotinit file describing the plotter must be available in one of the following:

- your home directory
- $install_dir/tools/plot/.cdsplotinit
- $cwd/ .cdsplotinit
- $HOME/ .cdsplotinit

A sample .cdsplotinit file is available in

your_install_dir/tools/plot/samples/cdsplotinit.sample

Two sample template files are available in

your_install_dir/tools/dfII/samples/plot/schPlot.il
your_install_dir/tools/dfII/samples/plot/schMetPlot.il

Arguments

t_file	A plot template file that stores plot options in a disembodied property list named schPlotOptions; must be enclosed in quotation marks.
w_windowId	Window ID in which the schematic is backannotated.

Value Returned

`t`	Generated a plot.
`nil`	Could not generate plot.

Examples

From window 2, generates the plot using the `plotTemplate` template file with backannotated values.

```
schPlot( "plotTemplate" window(2))
```

Related Topics

Plotting Designs

schProduceCVHierConfig

```
schProduceCVHierConfig(
     s_visitCB
     h_configId
     [ h_pathVectId ]
     )
     => t / nil
```

Description

Traverses a configuration design hierarchy and invokes a callback for each unique cellview.

When invoked, the *s_visitCB* procedure is passed with the arguments *d_cellViewId*, *x_depth*, *l_hierPath*, and *g_inStopCellView*.

Where,

- *d_cellViewId* is the cellview being visited.
- *x_depth* is the cellview depth in the hierarchy.
- *l_hierPath* is a list of cellviews in the hierarchical path.
- *g_inStopCellView* is a Boolean to indicate whether the visited cellview is in the viewStopList.

Arguments

s_visitCB	Name of the procedure to be invoked for each cellview.
h_configId	HDB configuration object identifier.
h_pathVectId	HDB path vector identifier.

Value Returned

t	The hierarchy was successfully traversed.
nil	No hierarchy traversal was performed.

Examples

```
(procedure (CCSPrintConfigSchematicViews @optional (win (hiGetCurrentWindow)))
  (procedure (visit cv depth hierPath inStopList)
    (for i 0 depth (printf ">"))
    (printf " %s.%s:%s\n" cv->libName cv->cellName cv->viewName))
  (let ((config (deGetConfigId win)))
    (when config (schProduceCVHierConfig 'visit config))))

(CCSPrintConfigSchematicViews)
```

schProduceUniqueCVHier

```
schProduceUniqueCVHier(
     s_visitCB
     d_topCellView
     g_viewNameList
     [ g_viewStopList ]
     )
     => t / nil
```

Description

Traverses a design hierarchy and invokes a procedure callback for each unique cellview.

When invoked, the *s_visitCB* procedure is passed with the arguments *d_cellViewId*, *x_depth*, *l_hierPath*, and *g_inStopCellView*.

Where,

- *d_cellViewId* is the cellview being visited.
- *x_depth* is the cellview depth in the hierarchy.
- *l_hierPath* is a list of cellviews in the hierarchical path.
- *g_inStopCellView* is a Boolean to indicate whether the visited cellview is in the viewStopList.

Arguments

s_visitCB	Name of the procedure to be invoked for each cellview.
d_topCellView	The top cellview from where traversal should begin.
g_viewNameList	An ordered list of view names to bind to during traversal (either a string or list of strings).
g_viewStopList	A list of view names at which to stop traversal (either a string or list of strings).

Value Returned

t	The hierarchy was successfully traversed.
nil	No hierarchy traversal was performed.

Examples

```
(procedure (CCSFindSchematicViews @optional (win (hiGetCurrentWindow)))
  (let (res)
    (procedure (visit cv depth hierPath inStopList)
      (setq res (cons cv res)))
    (schProduceUniqueCVHier 'visit win->topCellView "schematic symbol" "symbol")
    res))

(procedure (CCSCellViewName cv)
  (lsprintf "%s.%s:%s" cv->libName cv->cellName cv->viewName))

(foreach cv (CCSFindSchematicViews)
   (printf "%s\n" (CCSCellViewName cv)))
```

schRegisterCheckGroup

```
schRegisterCheckGroup(
     [ ?name s_name ]
     [ ?title t_title ]
     [ ?description t_description ]
     [ ?enabled g_enabled ]
     )
     => r_checkgroup / nil
```

Description

Creates and registers a new custom schematic checker group. Each new group appears as a discrete tab in the Schematic Rules Checks Setup form. Calling the function with the same group name as an existing group overwrites that group and all its rules.

Arguments

`?name` *`s_name`*

Symbolic name of the group which must be referenced when registering checks.

`?title` *`t_title`*

Tab title in the user interface. The default value is the *`s_name`* text.

`?description` *`t_description`*

Description for the custom schematic checker group.

`?enabled` *`g_enabled`*

Specifies whether the rules in this group should be run by default. Disabled groups will be shown grayed out in the user interface. The default value is `t`.

Value Returned

`r_checkgroup`	The new `schCheckGroup` structure was created and appears in the association list returned.
`nil`	The `schCheckGroup` structure could not be created.

Examples

```
(schRegisterCheckGroup
    ?name         'ercChecks
    ?title        "ERC"
    ?description  "Electrical Rule Checks"
    ?enabled      t)
=> schCheckGroup@0xfea8b98
```

Related Topics

Creating Custom Checks

schRegisterCheckRule

```
schRegisterCheckRule(
     [ ?title t_title ]
     [ ?name s_name ]
     [ ?groupName s_groupName ]
     [ ?checkCB u_checkCB ]
     [ ?configCB u_customCB ]
     [ ?severity s_severity ]
     [ ?description t_description ]
     )
     => r_checkrule / nil
```

Description

Creates and registers a new schematic rule checker. Calling this function with the same rule name as an existing rule in the same group overwrites that rule.

Arguments

`?title` *t_title*	Title text that for labels in the user interface.
`?name` *s_name*	Symbolic name of the created checker.
`?groupName` *s_groupName*	Symbolic name of the parent checker group.
`?checkCB` *u_checkCB*	The callback invoked on each run of the checker if the severity is not set as `'ignored`.
`?configCB` *u_configCB*	The callback invoked when clicking the "..." button in the user interface.
`?severity` *s_severity*	The default severity for this check (`'ignored` `'warning` `'error`).
`?description` *t_description*	Text for tooltips in the user interface.

Value Returned

r_checkrule	The new `schCheckRule` structure was created.
`nil`	The new `schCheckRule` structure could not be created.

Examples

```
(schRegisterCheckRule
    ?title        "No Shorting Transistors"
    ?name         'noTxShorts
    ?groupName    'ercChecks
    ?checkCB      'noTxShortsCB
    ?severity     'warning
    ?description  "An nfet/pfet transistor can not have the source connected to a
ground\nnet when the drain is connected to a power net")
=> schCheckRule@0x1154a828
```

Related Topics

Creating Custom Checks

schReportCheckFailure

```
schReportCheckFailure(
     [ ?object d_object ]
     [ ?checkRule r_checkRule ]
     [ ?message t_message ]
     [ ?short t_short ]
     )
     => t / nil
```

Description

Report the failure of a custom schematic check rule.

Arguments

`?object` *`d_object`*	The DB object to which the failure marker is to be attached. The default value is `d_cellView`.
`?checkRule` *`r_checkRule`*	The `schCheckRule` struct associated with this check.
`?message` *`t_message`*	The failure message to be reported.
`?short` *`t_short`*	The marker short message.

Value Returned

`t`	Successful failure report generated.
`nil`	Unsuccessful (for example, if `d_cellView` is not writable and `d_object` is provided).

Examples

```
(schReportCheckFailure
?object cv
?checkRule 'noTxShorts
?message "An nfet/pfet transistor can not have the source connected to a ground net
when the drain is connected to a power net")
=> t
```

schRegisterFixedMenu

```
schRegisterFixedMenu(
     t_category
     r_menuHandle
     [ s_disableTrigger ]
     [ s_enableTrigger ]
     )
     => t / nil
```

Description

Registers a vertical fixed-menu handle for specific cellview types to customize the schematic and symbol editor fixed menus. This function also registers triggers to enable and disable specific items in the fixed menu based on whether the editor window is in edit or read mode.

To reinstate the system default icon bar associated with the specified menu type, use the `schUnregisterFixedMenu` function.

A sample file containing the SKILL source code for the default fixed menus is located at *your_install_dir*`/tools/dfII/samples/local/schFixMenu.il`.

Arguments

t_category	Cellview type for which the menu is assigned; must be enclosed in quotation marks. Valid Values: `schematic`, `sheetSchematic`, `indexSchematic`, `symbol`
r_menuHandle	Menu to display. The `hiCreateVerticalFixedMenu` function creates *r_menuHandle*.
s_disableTrigger	SKILL function that is called when the cellview is opened in read mode or changed from edit to read mode. The trigger passes the *r_menuHandle* value of the fixed menu and the ID of the window containing the fixed menu. It calls `hiDisableMenuItem` for all menu entries to be disabled in read mode. Must be preceded by a tic mark (').
s_enableTrigger	SKILL function that is called when the cellview is opened in edit mode or changed from read to edit mode. The trigger passes the *r_menuHandle* value of the fixed menu and the ID of the window containing the fixed menu. It calls `hiEnableMenuItem` for all menu entries that might have been disabled by the disable trigger.

Value Returned

`t`	Registered a vertical fixed-menu handle for specific cellview types to customize the schematic and symbol editor fixed menus.
`nil`	Registering fixed-menu handle was unsuccessful.

Examples

Registers `myFixedMenu` as the fixed menu to display when the current cellview is a schematic.

```
schRegisterFixedMenu( "schematic" myFixedMenu )
```

Registers `symFixMenu` as the fixed menu to be displayed when the current cellview is a symbol. Also registers `symDisableProc` as the SKILL procedure to call when the cellview is in read mode and `symEnableProc` as the SKILL procedure to call when the cellview is in edit mode.

```
schRegisterFixedMenu( "symbol" symFixMenu 'symDisableProc 'symEnableProc )
```

Related Topics

Cadence SKILL Language Reference

schRegisterPopUpMenu

```
schRegisterPopUpMenu(
     t_category
     r_menuHandle
     t_mode
     )
     => t / nil
```

Description

Replaces the object-sensitive menus (OSMs) of the schematic canvas with a specific object category and edit mode. This function should only be called after a schematic cellview has been opened and the schematic editor has been initialized. To ensure this, call `schRegisterPopUpMenu()` from a callback registered using `deRegUserTriggers()`.

```
schRegisterPopUpMenu(
  t_category
  r_menuHandle
  t_mode
  )
  => t / nil
```

Arguments

t_category	Object type category to be replaced. Valid values: `"wire", "instance", "instPin", "label", "marker", "schNone", "schPin", "schMultiple", "schUnknown", "shapes", "symPin", "symNone", "symUnknown", "symMultiple", "schStandard", "symStandard", "indexDefault", "indexInstPin", "indexPin", "indexSheet", "border".`
r_menuHandle	Specifies the menu to be displayed. *r_menuHandle* is created by `hiCreateMenu`, as documented in the *Cadence SKILL Language Reference*.
t_mode	Access mode for which *r_menuHandle* is registered; must be enclosed in quotation marks. If you do not specify the mode, both modes are reassigned. Valid Values: `view`, `edit`

Value Returned

`t`	Registered *`r_menuHandle`* for a specific object type to customize the OSMs.
`nil`	Replacement was unsuccessful.

Examples

Registers `newInstanceMenu` as the OSM to display when the cursor is over an instance when the current cellview is opened in read mode.

```
schRegisterPopUpMenu( "instance" newInstanceMenu "view" )
```

Registers `newInstanceMenu` as the OSM to display when the cursor is over an instance when the current cellview is opened in edit mode.

```
schRegisterPopUpMenu( "instance" newInstanceMenu "edit" )
```

The following table shows object type selections.

ObjType (Category)	Selection
`instance`	instances
`schPin`	schematic pins
`instPin`	schematic instance pins
`wire`	wire (narrow) or wire (wide)
`label`	labels
`marker`	rectangle with layer
`schNone`	nothing under the cursor for schematic cellview
`schMultiple`	more than one object under cursor
`symPin`	symbol pins
`shapes`	symbol shapes
`symNone`	nothing under cursor for symbol cellview
`symMultiple`	more than one object under cursor
`symUnknown`	unknown figure under cursor
`schStandard`	schematic pop-up for nonsensitive

ObjType (Category)	Selection
symStandard	symbol pop-up for nonsensitive
indexSheet	instance of sheet in index cellview
indexPin	pin in index schematic
indexInstPin	instance pin in index schematic
indexDefault	any other object in index schematic
border	sheet border in schematic cellview

```
procedure( CDNCreateAndRegisterObjectMenus()
  let((item1 item2 menu)

    item1 = hiCreateMenuItem(
              ?name 'itemOne
              ?itemText "One"
              ?callback "println(1111)"
            )

    item2 = hiCreateMenuItem(
              ?name 'itemTwo
              ?itemText "Two"
              ?callback "println(2222)"
            )

    menu = hiCreateMenu(
             'MyInstanceMenu
             "My Instance Menu"
             list(item1 item2))

    schRegisterPopUpMenu("instance" menu "edit")
  )
)

deRegUserTriggers("schematic" nil nil 'CDNCreateAndRegisterObjectMenus)
```

Related Topics

Cadence SKILL Language Reference

schRegPostCheckTrigger

```
schRegPostCheckTrigger(
     s_functionName
     [ g_onceOnly ]
     )
     => t / nil
```

Description

Registers a function that will be called after a schematic is checked using the *Check – Current Cellview* or *Check – Hierarchy* commands. The called function must be defined to accept three arguments: the cellview ID of the schematic, the number of errors encountered, and the number of warnings encountered during the check.

If your registered function implements additional checks, consider also using `schUpdateUserSRCErrorAndWarn`.

Note: This function is called only on schematics with connectivity that requires checking.

There are limitations to be aware of when using `schRegPostCheckTrigger`:

- `schRegPostCheckTrigger` does not update the last checked timestamp.

  Note: If adding markers with a custom SKILL program, you can update the last checked timestamp using the `dbSetConnCurrent()` API, but only after checking that the connectivity was up to date when the trigger was called and the connectivity was not modified by the trigger (see the example below).

- If the trigger function calls any SKILL functions that modify the schematic database, you must ensure that these functions do not break the connectivity of the design.

Arguments

s_functionName	The symbol for the SKILL function that is to be called.
g_onceOnly	■ If not specified or specified as `'nil`, the registered function is called after each schematic is checked. ■ If any other value is specified, the registered function is only called once for the top schematic in the hierarchy; this allows you to register a routine that is called only once after a hierarchical check is performed, rather than for each schematic that is checked.

Value Returned

`t`	Registers a function that will be called after a schematic is checked.
`nil`	Registration was unsuccessful.

Examples

Registers `userSRC` as a function to be called after the specified cellview is checked.

```
procedure( userSRC(cv nErr nWarn "dxx")
    let( (current (nUserErr 0) (nUserWarn 0))
        printf( "Running userSRC ...\n" )


        ; Remember if connectivity is up to date.
        current = dbIsConnCurrent(cv)


        ; Create markers according to your schematic rule checks. Update the
        ; local variables nUserErr and nUserWarn accordingly.
        ; (code omitted)


        ; Include additional errors and warnings in the check report totals.
        schUpdateUserSRCErrorAndWarn(nUserErr nUserWarn)


        when(current && nUserErr == 0
        ; Zero errors, connectivity remains up to date.
        dbSetConnCurrent(cv))
      )
)


schRegPostCheckTrigger( 'userSRC )
```

Related Topics

schUpdateUserSRCErrorAndWarn

schRegPreCheckTrigger

```
schRegPreCheckTrigger(
     s_functionName
     [ g_onceOnly ]
     )
     => t / nil
```

Description

Registers a function that will be called before a schematic is checked using the *Check – Current Cellview* or *Check – Hierarchy* commands. The called function must be defined to accept one argument; that is the cellview ID of the schematic.

Note: This function is only called on schematics with connectivity that requires checking.

Arguments

s_functionName	The symbol for the SKILL function that is to be called.
g_onceOnly	■ If not specified, or specified as `nil`, the registered function is called before each schematic is checked. ■ If any other value is specified, the registered function is only called once for the top schematic. This allows you to register a routine that is called only once before a hierarchical check is performed, rather than for each schematic that is checked. **Note:** Each routine can only be registered once.

Value Returned

`t`	Registers a function that will be called before a schematic is checked.
`nil`	Registration was unsuccessful.

Examples

Registers `checkTrig` and `checkOnceTrig` as functions to be called. It then calls the pre-check trigger for schematics that require updating and automatically saves the schematics as they are checked.

```
procedure( checkTrig(cv "d")
    info(".. Check Trigger called for %s/%s/%s\n" cv~>libName cv~>cellName
cv~>viewName)
)

procedure(checkOnceTrig(cv "d")
    info(".. Check Once Trigger called for %s/%s/%s\n" cv~>libName
cv~>cellName cv~>viewName)
)

schRegPreCheckTrigger 'checkTrig
schRegPreCheckTrigger 'checkOnceTrig t

envSetVal("schematic" "checkAlways" 'boolean nil)
envSetVal("schematic" "checkHierSave" 'boolean t)

schCheckHier( geGetEditCellView() "schematic" " " )
```

Calls the pre-check trigger for every schematic in the hierarchy. The modified schematics are not saved.

```
envSetVal("schematic" "checkAlways" 'boolean t)
envSetVal("schematic" "checkHierSave" 'boolean nil)

schCheckHier( geGetEditCellView() "schematic" " " )
```

Related Topics

schUnregPreCheckTrigger

schReinitializeShapeStyles

```
schReinitializeShapeStyles(
     [ ?displayName t_displayName ]
     )
     => t / nil
```

Description

Updates the style of all schematic shapes in the current session to reflect any changes made in the Display Resource Editor. Use this function to refresh non-default shape styles, for example, custom wire line styles and colors.

Arguments

`?displayName` *`t_displayName`*

The current display name. Specifies on-screen or printable format.
Default: `display`

Value Returned

`t`	Command completed successfully.
`nil`	Command failed.

Examples

```
schReinitializeShapeStyles ( ?displayName "display" )
=> t
```

schRemoveIgnoreProp

```
schRemoveIgnoreProp(
     t_name
     )
     => t / nil
```

Description

Removes the specified ignore property from the registered ignore property set. You can view the registered ignore properties in the *Ignore Properties* tab by clicking *Options – Editor*.

Arguments

t_name	Name of the property to be removed from ignore property set, such as `ignore`, `lvsIgnore`, `nlAction`, or `nlIgnore`.

Value Returned

`t`	Property is removed successfully.
`nil`	Unable to remove the ignore property.

Examples

Returns `t` if removal is successful. If the property does not exist in the registered ignore property set, the function returns `nil`.

```
schRemoveIgnoreProp( "nlAction" )
```

schRenumberAllSheet

```
schRenumberAllSheet(
     d_cvId
     )
     => t / nil
```

Description

Resequences all sheets starting at 1 and fills any holes in the sequence.

Arguments

d_cvId	Cellview ID of the index for a multisheet schematic design.

Value Returned

t	Resequenced all sheets starting at one and fills any holes in the sequence.
nil	Resequencing was unsuccessful.

Examples

Resequences the sheets contained in the index schematic cellview.

```
schRenumberAllSheet( cv )
```

schRenumberInstances

```
schRenumberInstances(
     g_objId
     [ t_scope ]
     [ g_verbose ]
     [ t_sequence ]
     [ x_startIndex ]
     [ t_applyTo [ t_libraryName t_cellName t_viewName ] ]
     )
     => t / nil
```

Description

Resequences instances using the format *instNamePrefix number* that results in unique numbering indexes for each component name prefix encountered. Any voids in a numbering sequence are resolved by renaming instances with the highest numbers to fill the voids.

Arguments

g_objId	ID of a cellview or library. Valid Values: db `cellview ID` (*d_cvId*), `dd library` (*b_libId*)
t_scope	Defines the range of cellviews to have their instances renumbered; must be enclosed in quotation marks. Valid Values: For a cellview ID: `cellview`, `hierarchy`; for a library ID: `library` Default: `cellview` when the ID is *d_cvId*; `library` when the ID is *b_libId*
g_verbose	Echoes the renumbered instance names and source names to the CIW. Default: `nil`
t_sequence	Defines the sequencing mechanism; must be enclosed in quotation marks. Valid Values: `filling the voids`, `X+Y+`, `Y+X+`, `X+Y-`, `Y-X+`, `X-Y+`, `Y+X-`, `X-Y-`, `Y-X-` Default: `filling the voids` **Note:** `X` + `Y` renumbers from the bottom left to the top right, going left to right, bottom to top.
x_startIndex	The number assigned to the first renumbered instance.

t_applyTo	Defines the group of instances; must be enclosed in quotation marks. To renumber all instances, specify `any master`; to renumber only the instances of a specific master, specify `same master` and specify *t_libraryName t_cellName t_viewName* Valid Values: `any master`, `same master` Default: `any master`
t_libraryName t_cellName t_viewName	The library, cell, and view name of the instance to apply the renumbering sequence to; each item must be enclosed in quotation marks.

Value Returned

`t`	Resequenced instances using the specified format.
`nil`	Instances were not resequenced.

Examples

Resequences the instances contained in the cellview.

```
schRenumberInstances( cv )
```

Resequences the `symbol` view of the cell `buf` from the library `sample` with verbose set to on, using the Y-X+ order, starting the sequence with index 0.

```
schRenumberInstances( cv "cellview" t "Y-X+" 0 "same master" "sample" "buf"
"symbol" )
```

schRenumberSheet

```
schRenumberSheet(
     d_cvId
     x_from
     x_to
     )
     => t / nil
```

Description

Changes the number of a sheet in a multisheet schematic and changes the cell name of the renumbered schematic to match the destination sheet number. If a sheet already exists with the destination number, the new sheet is inserted before it and all succeeding sheets are renumbered accordingly.

Arguments

d_cvId	Cellview ID of the index schematic containing the sheet to renumber.
x_from	Source number of the sheet to renumber.
x_to	Destination number for the source sheet.

Value Returned

t	Changed the number of a sheet in a multisheet schematic and changes the cell name of the renumbered schematic to match the destination sheet number.
nil	Changing sheet number was unsuccessful.

Examples

Renumbers sheet 3 as sheet 4. Any succeeding sheets are renumbered.

```
schRenumberSheet( cv 3 4 )
```

Renumbers sheet 4 as sheet 3. If sheet 3 exists, the sheets are swapped.

```
schRenumberSheet( cv 4 3 )
```

schReplaceProperty

```
schReplaceProperty(
     l_objId
     t_propName
     t_propValue
     )
     => t / nil
```

Description

Changes the value of *t_propName* to *t_propValue* for the object. This function checks if the net, pin terminal, and master properties exist for the object.

This function can replace only those properties that can be modified with dbSetq.

Arguments

l_objId	List that contains one object ID. *l_objId* must be a database object in a schematic or symbol cellview. The cellview containing the property must be editable.
t_propName	Name of the property to add or modify; must be enclosed in quotation marks.
t_propValue	String value of the property to assign; must be enclosed in quotation marks. The value type is converted to the found property. If no property is found, a property of type string is created. If *t_propName* is master, *t_propValue* must be a string and the string must contain *t_libName t_cellName t_viewName* (separated by spaces).

Value Returned

`t`	Changes the value of *`t_propName`* to *`t_propValue`* for the object.
`nil`	Could not change the value.

Examples

Changes the instance name of specified instance to `myInst`.

```
instList = list( instId )
schReplaceProperty( instList "instName" "myInst" )
```

schSaveCurrentPlotOptions

```
schSaveCurrentPlotOptions(
     t_fileName
     )
     => t / nil
```

Description

Writes the current plot settings to a file.

Arguments

t_fileName	The filename where you want to save the settings.

Value Returned

t	Wrote the current plot settings to the file which you specified.
nil	Unsuccessful.

Examples

Writes the current plot settings to `myPlotSettingsFile`. You can add this function to your `.cdsinit` file to ensure that every time you exit the software, the last plot settings are automatically saved in the specified file.

```
myPlotSettingsFile="userHomeDir/currentPlotSettings"

procedure( RememberPlotSettings()
     schSaveCurrentPlotOptions( myPlotSettingsFile )
)

regExitBefore( 'RememberPlotSettings )
```

schSchemToPinList

```
schSchemToPinList(
     t_libName
     t_cellName
     t_viewName
     )
     => g_pinList
```

Description

Generates a pin list from a schematic cellview.

Arguments

t_libName	Library containing the schematic; must be enclosed in quotation marks.
t_cellName	Cell containing the schematic; must be enclosed in quotation marks.
t_viewName	View containing the schematic; must be enclosed in quotation marks.

Value Returned

g_pinList	Terminal and property information in the form of a pin list, generated from the source schematic.

Examples

Returns the pin list representing the source schematic.

```
pinList = schSchemToPinList( "myLib" "myDesign" "schematic" )
```

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '( nil ports portList
         [prop proplist] )
portlist = ( termDef termDef...termDef )
termDef = ( nil name "termName"
         direction termDir
         [pins termPins]
         )
proplist = ( nil propName propValue
```

```
    propName propValue
    ...
    )
termPins = ( pinDef pinDef...pinDef )
pinDef =  ( nil name "pinName"
    [accessDir "accessDir"] )
```

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for usage of schSchemToPinList in the schViewToPinListReg list of translation functions.

schSelectAllFig

```
schSelectAllFig(
     [ d_cvId ]
     )
     => t
```

Description

Selects all objects in a cellview that pass the selection filter.

Arguments

d_cvId	Cellview ID of the cellview you want to select. If not specified, the current cellview is used.

Value Returned

Always returns `t`.

Examples

Selects all objects from the cellview in the current window.

```
schSelectAllFig( )
```

schSelectPoint

```
schSelectPoint(
     w_windowId
     l_pt
     g_isPartial
     g_isAdditive
     x_timeDelay
     )
     => t / nil
```

Description

Interactively selects the object under the cursor. With single selection, this function first deselects all objects on the selected set. With additive selection, this function maintains the selected set and adds the current object to the selected set.

These procedures have the same functionality as `mouseSingleSelectPt` and `mouseAddSelectPt` as defined by the schematic editor.

This function also sets the most-recently selected object needed by extended selection. If time has not expired (as defined by *x_timeDelay*), this function calls extended selection instead of simple selection.

You can use this function only for schematics.

Arguments

w_windowId	Window in which to apply selection.
l_pt	List of X and Y coordinates that define the selection area.
g_isPartial	Boolean flag that specifies if partial selection is supported.
g_isAdditive	Boolean flag that specifies if selection is single or additive.
x_timeDelay	Specifies how much time must elapse before the selection becomes simple selection. If the command is executed a second time before time has elapsed, extended selection is applied.

Value Returned

`t`	Interactively selects the object under the cursor.
`nil`	Selection was unsuccessful.

Examples

Deselects all objects. If the cursor is over an object, the object is selected.

```
schSelectPoint( hiGetCurrentWindow( ) hiGetCommandPoint( ) t nil 0)
```

If the cursor is over the object, the object is added to the selected set.

```
schSelectPoint( hiGetCurrentWindow( ) hiGetCommandPoint( ) nil t 0 )
```

schSetAndLoadTsgTemplateType

```
schSetAndLoadTsgTemplateType(
     t_tsgType
     [ t_templateFileName ]
     )
     => t / nil
```

Description

Sets the `tsgTemplateType` environment variable and loads the corresponding `tsg` template file.

Use this function to load a new TSG template file to overwrite the currently loaded TSG template file. A TSG template file is automatically loaded when you first create a symbol or first open the Symbol Generation Options form. After that time, a TSG template file is only loaded upon request.

`schSetAndLoadTsgTemplateType` is the procedural equivalent to the *Load* button of the Symbol Generation Options form.

A TSG template file contains settings that describe the attributes, labels, and properties that the `tsg` engine references when creating symbols automatically.

Arguments

t_tsgType

Sets the `tsgTemplateType` environment variable to this keyword. This keyword indirectly references the full path to a TSG template file using the `tsgTemplatesMasters` list, which is defined in the `schConfig.il` file.

The Cadence-provided keywords and TSG template files are as follows:

Keyword	Full Path to tsg Template File
digital	*your_install_dir*`/tools/dfII/samples/symbolGen/default.tsg`
artist	*your_install_dir*`/tools/dfII/samples/symbolGen/artist.tsg`
PCB	*your_install_dir*`/tools/dfII/samples/symbolGen/package.tsg`
metric	*your_install_dir*`/tools/dfII/samples/symbolGen/metric.tsg`
other	not defined

t_templateFileName

If you specify `other` as the *t_tsgType* keyword, this argument lets you specify the full path to any TSG template file rather than using the `tsgType` keyword mapping described above.

Value Returned

`t`	Set the `tsgTemplateType` environment variable and performs a load of the corresponding tsg template file.
`nil`	Unsuccessful.

Examples

Assigns the keyword `metric` to the `tsgTemplateType` environment variable and reads the settings from the corresponding `tsg` template file. The system references those settings when creating symbols automatically.

```
schSetAndLoadTsgTemplateType( "metric" )
```

Related Topics

Symbol Generation Options Form

TSG template file

schSetBundleDisplayMode

```
schSetBundleDisplayMode(
     d_labelId
     t_displayMode
     )
     => t / nil
```

Description

Takes a label Id whose display mode is to be changed and applies either a `vertical` or `horizontal` display value. This will only work when the cellview is editable.

Arguments

d_labelId	Label Id whose bundle display mode is to be set.
t_displayMode	Sets the label Id display mode to be either `vertical` or `horizontal`.

Value Returned

`t`	Successfully set bundle display mode value.
`nil`	Unsuccessful. Failure can occur if the Id specified is not a label or if an incorrect value is set for the display mode.

Examples

If `labId` represents a wire bundle label which is being displayed horizontally, but you want to change this to vertical, then:

```
schSetBundleDisplayMode (labId "vertical") => vertical
```

If `figId` represents the Id of a pin name, then:

```
schSetBundleDisplayMode (figId "vertical") => nil
```

schSetCmdOption

```
schSetCmdOption(
     g_form
     s_field
     l_fieldValues
     x_key
     t_mousePrompt
     )
     => t / nil
```

Description

Customizes which form fields are modified by calls to `schCmdOption`, middle mouse button, and `schShiftCmdOption`, `Shift`-middle mouse button, when the command is active.

Arguments

g_form	Form to customize.
s_field	Symbol of the form field to modify.
l_fieldValues	List of valid values that the field cycles through.
x_key	Specifies whether this form field is changed during `schCmdOption` or `schShiftCmdOption`. Valid Values: `1` for normal, `2` for shift
t_mousePrompt	Description of the command that will be displayed on the status line to explain the effect of clicking on the mouse button; must be enclosed in quotation marks.

Value Returned

`t`	Customized which form fields are modified by calls to `schCmdOption`, middle mouse button, and `schShiftCmdOption`, `Shift`-middle mouse button, when the command is active.
`nil`	Customization was unsuccessful.

Examples

Sets the direction field to toggle between `input` and `output` when the `createPin` command is active and you click the right mouse button.

```
schSetCmdOption( schCreatePinForm 'direction list( "input" "output" ) 1 "toggle
Direction" )
```

schSetEnv

```
schSetEnv(
     t_variableName
     g_value
     )
     => t / nil
```

Description

Sets the value of a schematic environment variable.

This function, along with the `schGetEnv` function, lets you program the values for various options within the schematic editor without using a form. Also, these functions complement the general environment variable mechanism, which lets you preset values at startup using a `.cdsenv` file.

Arguments

t_variableName	Name of the schematic environment variable whose value you want to set.; must be enclosed in quotation marks.
g_value	The value to give the variable. This varies depending on the variable. Refer to *Virtuoso Schematic Editor User Guide* for environment variable descriptions.

Value Returned

`t`	Set the value of a schematic environment variable.
`nil`	Either the named variable is not a schematic environment variable or the value is of the wrong type.

Examples

Sets the value of the `maxLabelOffsetUU` schematic environment variable to `0.0125`. This value is then used by both the schematic extractor and the schematic rules checker.

```
result = schSetEnv( "maxLabelOffsetUU" 0.0125 )
```

Sets the value of the `vicViewList` environment variable for the cross-view-checker.

```
schSetEnv( "vicViewList" "layout symbol" )
```

Related Topics

schGetEnv

schSetIgnorePropEnabled

```
schSetIgnorePropEnabled(
     t_name
     g_enable
     )
     => t / nil
```

Description

Enables or disables a particular ignore property. When you enable an ignore property, this property is applied on an instance while ignoring that instance.

Arguments

t_name	Name of the property that needs to be enabled or disabled.
g_enable	Enable or disable the property by specifying `t` or `nil`.

Value Returned

`t`	The property was enabled or disabled successfully.
`nil`	Operation was unsuccessful.

Examples

Suppose ignore property `nlIgnore` needs to be enabled:

```
schSetIgnorePropEnabled(?name "nlIgnore" ?enable t)
```

For disabling `nlIgnore`:

```
schSetIgnorePropEnabled(?name "nlIgnore" ?enable nil)
```

schSetOrigin

```
schSetOrigin(
     d_cvId
     l_origin
     )
     => t / nil
```

Description

Repositions all the objects in the specified symbol or schematic cellview relative to the new point.

Arguments

d_cvId	Cellview ID of an editable symbol or schematic cellview where you set the origin for repositioning the objects.
l_origin	Origin specified as a point.

Value Returned

t	Repositioned all the objects in the specified cellview relative to the new origin point.
nil	Operation was unsuccessful.

Examples

This example shows how to move the origin of the current cellview to point (1, 0).

```
schSetOrigin( geGetEditCellView() 1:0 )
```

schSetPropertyDisplay

```
schSetPropertyDisplay(
     ?object d_object
     [?name S_name]
     ?visibility S_visibility
     )
     => l_textDisplays / nil
```

Description

Shows or hides textDisplays containing the name and value of the attribute, property, or parameter of an object. TextDisplay locations are chosen automatically to avoid overlaps with other textDisplays. For more information, see schGetPropertyDisplay and dbCreateTextDisplay.

Arguments

`?object` *d_object*	The object whose attribute, property, or parameter you want to display. Supported objects are: `instance`, `textDisplay`.
`?name` *S_name*	Name of the attribute, property, or parameter of `d_object` whose display characteristics you want to change. The supported values depend on the type of `d_object`. Refer to the following `t_name` values for `d_object`. **d_object type - t_name** **Instance -** `libName`, `cellName`, `viewName`, `name`; The name of any property on the instance; the name of any CDF parameter on the instance; `nil` to manipulate all textDisplays owned by the instance. **textDisplay** - `nil` The textDisplay's name and value settings can be changed directly by `s_visibility`.
`?visibility` *S_visibility*	

Values for this argument:

`'off` hides the element from view, but does not delete it

`'name` shows only the name of the element, for example, *`cellName`*

`'value` shows only the value of the element, for example, *`nmos`*

`'both` shows both the element name and value, for example, *`cellName=nmos`*

Note: `'off` does not delete a textDisplay. If you subsequently make it visible, it appears at its previous location. To completely delete a textDisplay, use schDelete.

Return Value

`l_textDisplays`	A list of existing or newly created textDisplay objects that display the requested property. Usually, there is only one such textDisplay.
`nil`	If the requested property could not be displayed.

Examples

Displays the favorite items for an instance

```
schSetPropertyDisplay(?object inst
                    ?name "name"
                    ?visibility 'both)
schSetPropertyDisplay(?object inst
                    ?name "cellName"
                    ?visibility 'value)
schSetPropertyDisplay(?object inst
                        ?name "vdd"
                        ?visibility 'both)
schSetPropertyDisplay(?object inst
                    ?name "model"
                    ?visibility 'value)
```

Turns on all netSet properties for an instance.

```
foreach(netSet setof(prop
```

```
                    inst->prop
                    prop->valueType == "netSet")
schSetPropertyDisplay(?object inst
                    ?name netSet->name
                    ?visibility 'both)
)
```

Switches off all displayed properties on an instance

```
schSetPropertyDisplay(?object inst ?visibility 'off)
```

Permanently removes all `'off` displayed properties

```
foreach(td schGetPropertyDisplay(?object inst)
when(schGetPropertyDisplay(?object td) == 'off
schDelete(td)
)
)
```

Related Topics

schGetPropertyDisplay

dbCreateTextDisplay

schDelete

schSetShapeStyle

```
schSetShapeStyle(
     d_shape
     [ ?color t_colorName ]
     [ ?lineStyle t_lineStyleName ]
     [ ?stipple t_stippleName ]
     [ ?fillStyle t_fillStyleName ]
     [ ?fillColor t_colorName ]
     )
     => t / nil
```

Description

Specifies the style attributes of a shape to override the defaults provided by the Display Resource file. Each attribute value must be valid for the current display name.

Arguments

d_shape	A schematic symbol shape ID.
?color *t_colorName*	Name of any color defined in the Display Resource Editor.
?lineStyle *t_lineStyleName*	Name of line style defined in the Display Resource Editor.
?stipple *t_stippleName*	Name of stipple style defined in the Display Resource Editor.
?fillStyle *t_fillStyleName*	Name of the fill style. Valid values: ■ `outline` - use no shape fill and draw only the shape outline. ■ `solid` - fill the shape with a solid color and outline. ■ `X` - fill the shape with a cross hatched pattern. ■ `stipple` - fill the shape with a stipple pattern defined using `?stipple` but draw no shape outline. ■ `outlineStipple` - fill the shape with a stipple pattern specified using `?stipple` and draw a shape outline specified in `?outlineColor`.
?fillColor *t_colorName*	The shape fill color name as specified in the Display Resource Editor.

Value Returned

`t`	Returns `t` on success.
`nil`	One or more attributes specified are invalid.

Examples

Selects a shape on the canvas and passes it as an argument to `schSetShapeStyle`:

```
schSetShapeStyle(car(selectedSet() ?color "orange" ?lineStyle "dashed")
```

```
=> t
```

schSetSignalTypeIntegrity

```
schSetSignalTypeIntegrity(
     t_netSigType
     t_termNetSigType
     t_termDirection
     t_severity
     )
     => t / nil
```

Description

Sets the conflict severity for a signal type integrity check (available as part of Schematic Rule Checks) for connections between nets of a specified signal type, and instance pins of specified signal type and direction.

Arguments

t_netSigType	The specified signal type.
t_termNetSigType	The terminal net signal type.
t_termDirection	The terminal direction.
t_severity	The severity setting to be applied to the signal type (“ignored”, “warning” or “error”).

Value Returned

t	Signal type integrity successfully set.
nil	Command failed.

Examples

```
schSetSignalTypeIntegrity("tieOff" "scan" "inputOutput" "error")
==> t
```

Related Topics

schGetSignalTypeIntegrity

schSetSplitPrimaryInst

```
schSetSplitPrimaryInst(
     d_splitInstId
     g_primInstId
     )
     => t / nil
```

Description

Associates the given split instance with the given split-primary.

Note: The given instance is a valid split-primary for the split. For example, a regular instance cannot be declared a split-primary for a given split.

Arguments

d_splitInstId	The Split instance ID.
g_primInstId	The Split-primary instance ID.

Value Returned

t	Makes the specified instance a split-primary of the given split.
nil	Unsuccessful.

Examples

The following example creates a split instance with name I0_s1. This automatically sets up a split-primary, named I0, on the newly created instance. It then creates a copy of I0_s1. The copied instance does not have a split-primary. Therefore, a split-primary is set up for it.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "r")
i0_s1 = schCreateInst(cvId s1 "I0" 0:1 "R0")
i0 = schGetSplitPrimaryInst(i0_s1)
i1_s1 = schCopy(i0_s1 cvId list(2:0 "R0")
schGetSplitPrimaryInst(i1_s1)
=> nil
i1 = schCopy(i0 cvId list(0:0 "R0")
schSetSplitPrimaryInst(i1_s1 i1)
=> t
```

```
schGetSplitPrimaryInst(i1_s1)
=> [ Returns i1 ]
```

schSetSplitSymbol

```
schSetSplitSymbol(
     d_cvId
     [ g_isSplitSymbol ]
     )
     => t / nil
```

Description

Sets up special properties on the cellview to make it a split symbol of `g_isSplitSymbol` is `t`. However, if `g_isSplitSymbol` is `nil`, the specified cellview is converted to a normal symbol by stripping off the aforementioned properties.

Arguments

d_cvId	The ID of an editable cellview with viewType as `schematicSymbol`.
g_isSplitSymbol	Specifies whether or not to create the cellview as a split symbol. The default value is `t`.

Value Returned

`t`	The specified cellview is created as a split symbol.
`nil`	The specified cellview remains unchanged.

Examples

The following example marks a regular cellview `lib bga s1` of type `schematicSymbol` as a split symbol.

```
s1 = dbOpenCellViewByType("lib" "bga" "s1" "schematicSymbol" "a")
schSetSplitSymbol(s1)
dbSave(s1)
=> t
```

schSetSymbolOrigin

```
schSetSymbolOrigin(
     d_cvId
     l_origin
     )
     => t / nil
```

Description

Moves all the objects in the specified symbol cellview relative to the given origin point.

Arguments

d_cvId	Cellview ID of an editable symbol cellview in which to set the origin for moving the objects.
l_origin	Origin specified as a point.

Value Returned

t	Moved all the objects in the specified cellview relative to the given origin point.
nil	Move was unsuccessful.

Example

Moves the origin of the cellview specified by `symbolCV` to the point 1,0.

```
schSetSymbolOrigin( symbolCV 1:0 )
```

schSetTextDisplayBBox

```
schSetTextDisplayBBox(
     d_tdId
     d_instId
     )
     => t / nil
```

Description

Sets or updates the value of a bounding box that encloses a given `textDisplay` object. A text display object displays the string or value based on derived information; for example, the current value of a property. Accordingly, the software must update the bounding box that encloses the text display object when the derived string or value changes.

Arguments

d_tdId	ID of the text display object. A text display object has all the characteristics of a label, such as `fontStyle`, `fontHeight`, and `overBar`.
d_instId	ID of an instance when *d_tdId* is within the symbol cellview of the instance. You should set this argument to `nil` when *d_tdId* is in the current schematic.

Value Returned

`t`	Set or updated the value of a bounding box that encloses a given `textDisplay` object.
`nil`	Value setting or update was unsuccessful.

Examples

Sets the bounding box of the text display object as defined by `tdId`.

```
schSetTextDisplayBBox( tdId nil )
```

Sets the bounding box of the text display object as defined by both `tdId` and `instId`.

```
schSetTextDisplayBBox( tdId instId )
```

schSetWireColor

```
schSetWireColor(
     d_wireId
     t_colorName
     ) => t / nil
```

Description

Sets the color used to draw a wire segment.

Arguments

d_wireId	The wire segment ID.
t_colorName	The wire segment's new color. Colors are referred to by the names defined in the Display Resource File.

Value Returned

t	When the color of the wire is set.
nil	When the color of the wire is not set.

Examples

Here is how to change the color of all wire segments on a given net:

```
procedure(mySetNetColor(net colorName)
    foreach(fig net->figs
           schSetWireColor(fig colorName)
    )
    t
)
```

You could use it on the "vdd!" net in a schematic opened in VSE like this:

```
mySetNetColor(dbFindNetByName(geGetEditCellView() "vdd!") "red")
```

Related Topics

schGetWireColor

schGetWireLineStyle

schSetWireLineStyle

schCreateWire

schSetWireLineStyle

```
schSetWireLineStyle(
     d_wireId
     t_lineStyleName
     ) => t / nil
```

Description

Sets the line style used to draw a wire segment.

Arguments

d_wireId	The wire segment ID.
t_lineStyleName	The wire segment's new line style. Line styles are referred to by the names defined in the Display Resource File.

Value Returned

t	When the line style of the wire is set.
nil	When the line style of the wire is not set.

Examples

Here is how to change the line style of all wire segments on a given net:

```
procedure(mySetNetLineStyle(net lineStyleName)
       foreach(fig net->figs
           schSetWireLineStyle(fig lineStyleName)
       )
       t
   )
```

You could use it on the "vdd!" net in a schematic opened in VSE like this:

```
mySetNetLineStyle(dbFindNetByName(geGetEditCellView() "vdd!") "dots")
```

Related Topics

schGetWireLineStyle

schGetWireColor

schSetWireColor

schCreateWire

schShiftCmdOption

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

Description

Cycles through a predefined set of values. By default, this function is bound to the `Shift`-middle mouse button. When you click the middle mouse button while pressing the `Shift` key during an active command, the command applies the next value in the predefined set. You can customize the predefined set of values by making calls to `schSetCmdOption`.

Arguments

None

Value Returned

`t`	Cycled through a predefined set of values.
`nil`	Cycling was unsuccessful.

schSingleSelectBox

```
schSingleSelectBox(
     [ w_windowId ]
     [ g_partial ]
     [ l_bBox ]
     )
     => t
```

Description

Selects objects within a rectangular area from a specified schematic editing window. With no arguments, it prompts you to enter the area to be selected in the current window. Partial selection is performed if the window environment variable *partialSelect* is set.

Arguments

w_windowId	Database ID of the window containing the objects.
g_partial	Indicates whether partial selection should be performed.
l_bBox	List specifying the corners of the rectangular area to select. If not specified, or specified as `nil`, you are prompted to define the area with the mouse.

Value Returned

Always returns `t`.

Examples

Prompts you to define the area in the current window for selection with the mouse.

```
schSingleSelectBox( )
```

schSnapToConn

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

Description

Interactively connects a wire to the nearest connectivity object during the *Create Wire* command and the schematic `snapEnabled` environment variable is set.

Can only be used when the *Create Wire* command is active in the schematic editor.

Arguments

None

Value Returned

`t`	Interactively connects a wire to the nearest connectivity object during the *Create Wire* command and the schematic `snapEnabled` environment variable is set.
`nil`	Connecting was unsuccessful.

schSnapToGrid

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

Description

Places the object on the grid in a schematic or symbol view.

Arguments

d_figId	Figure ID of the object that needs to be placed on grid.

Value Returned

t	The object has been snapped to the grid successfully.
nil	Snapping was unsuccessful.

Examples

```
schSnapToGrid( anyObjectFigId )
```

Related Topics

Snapping To Grid

schSolder

```
schSolder(
     d_cvId
     l_solderPt
     )
     => d_shapeId / nil
```

Description

Solders two wires together in a schematic. This function places the solder dot over the given point if it is a + or T- wire intersection.

Arguments

d_cvId	Cellview ID of the cellview to contain the solder dot. Must be a schematic cellview.
l_solderPt	Location of the solder dot specified as a point.

Value Returned

d_shapeId	The ID of the specified shape.
nil	Solder dot placement unsuccessful.

Examples

```
shapeId = schSolder( cv 2:3 )
```

Places a solder dot at the specified location.

schSplitNumber

```
schSplitNumber(
     )
     => string
```

Description

Used as an IL label on a split symbol. It provides the split index (starting from 1) to the instance on which the label is attached. When split instances are added to a schematic, it is useful to have a label on the instance that indicates the index of the split out of the total number splits for the device.

Arguments

None

Value Returned

`string`	Returns a string in the format `x of y`, where `x` is the split index and `y` is the total number of split masters for the component.

Examples

When there are five split symbols for a device, adding the first split, `s1`, the label will display `1 of 5`.

```
schSplitNumber()
```

schSRC

```
schSRC(
     d_cvId
     )
     => l_result
```

Description

Runs the schematic rules checker (SRC) on a specified cellview.

You can set the schematic rules checker rules by

- Specifying options on the Setup Schematic Rules Checks options form
- Calling `schSetEnv` to set the schematic environment variable that controls a check
- Specifying values for the schematic environment variables in your `.cdsenv` file

You can set the values for the schematic environment variables that control the logical, physical, and name checks. For most of the schematic environment variables that control checks, the three possible values are `ignored`, `warning`, and `error`. These three values are collectively known as the check severity.

- When you set the check severity value for a variable to `ignored`, the system does not perform the check associated with that variable.
- When you set the check severity value for a variable to `warning`, the system marks any violations discovered during the check as warnings. You can save a design that contains warnings, and you can simulate a design that contains warnings. Nevertheless, you should review the warnings before proceeding.
- When you set the check severity value for a variable to `error`, the system marks any violations discovered during the check as errors. You can save a design that contains errors, but you cannot simulate the design simulation until you correct the errors.

Can be used only on an editable schematic.

Arguments

d_cvId	Cellview ID on which to run the schematic rules checker.

Value Returned

l_result	A list containing the number of errors and warnings, respectively.

Examples

Runs the schematic rules checker on the given cellview and extracts the number of errors and warnings from the result.

```
result = schSRC( cvId )
numErrors = car( result )
numWarns = cadr( result )
```

Sets the severity of the three checks—`srcUnconnectedWires`, `srcVerilogSyntax`, and `srcVHDLSyntax`—and then invokes the schematic rules checker.

```
schSetEnv( "srcUnconnectedWires" "ignored" )
schSetEnv( "srcVerilogSyntax" "error" )
schSetEnv( "srcVHDLSyntax" "ignored" )
result = schSRC( cvId )
```

schStretch

```
schStretch(
     l_figIds
     l_transform
     )
     => t / nil
```

Description

The `schStretch` command can be used to stretch objects. The API takes a list of `figIds` and a transformation, and then stretches the specified figure in `figId~>cellView` by applying the transformation.

Note: `schStretch` is a non-HI equivalent of `schHiStretch`.

The `schStretch` function has no requirement for a cellview to be open in a window, and can operate successfully when the cellview is opened using `dbOpenCellViewByType`.

The `schStretch` function also follows the same constrains as those followed by `schHiStretch` (for example, the error messages are the same and all environment variables are obeyed).

The stretching behavior performed is also similar to that used by `schHiStretch`, and you are able to control the stretch behavior using the environment variable `stretchMethod`.

There is however one important difference, where stretching is performed within the cellview of the `figId` (`figId~>cellview`). For instances (symbol and schematic), wires, markers, patch cords, pins, pin and wire names, and so on, that are placed on the scematic, the stretching is done in that schematic cellview. If however you specify the figId of some object that is in a symbol (figId of a symbol pin), that would be stretched inside a symbol. In this scenario, the behavior would be the same as `schMove`.

With object stretching, the reference point is taken as the origin. The `figIds` are stretched in their totality, as if in fully selected mode, in the same way that `schCopy` and `schMove` works.

Note: As the stretching reference point is always the origin, if you want a different origin, you will need to adhere to the following example:

If xy is the reference point that you want to use, and xform is the transformation relative to that reference point, then:

```
newXform=dbConcatTransform(
     dbConcatTransform(
     list(mapcar('minus xy) "R0" 1) xform)
     list(xy "R0" 1))
```

And then do:

```
schStretch(figs newXform)
```

Arguments

l_figIds	The list of ids of the objects to be stretched together. All objects must belong to the same cellview. Only objects in the schematic or schematicSymbol view types are stretched. The cellview type for stretching is determined by checking the cellview of the first object in the `l_figIds`.
l_transform	Specifies the relative location, orientation, and optionally magnification of the moved figure, specified as a list of the form: (*l_offset t_orient [n_magnification]*) Where: *l_offset* is the offset from the original position expressed as a list of two floats, the first specifying the distance to move in the x direction and the second the distance in the y direction; for example (10.0:5.0). *t_orient* specifies the orientation of the moved object and is one of R0, R90, R180, R270, MX, MXR90, MY, MYR90. The value must be enclosed in double quotes. *n_magnification* specifies the relative size of the moved object. The default is 1.0 (i.e. the same size as before the move).

Value Returned

`t`	Stretch operation has been successful.
`nil`	Stretch operation has been unsuccessful.

Examples

To use the `schStretch()` command, you can take the following steps:

1. Open a schematic that contains some instances.
2. Select the objects that you wish to stretch
3. In the CIW enter:

```
figIds = geGetSelSet()
trans = list (0:0 "R90")
schStretch (figIds trans)
```

This first of all gets a list of the selected figIds, rotates the instances by 90, before then performing the stretch.99

Alternatively you could:

1. Open a schematic that contains some instances.
2. In the CIW enter:

```
cv = geGetEditCellView()
figIds = cv~>instances
trans = list (-1:-1 "R90")
schStretch(figIds trans)
```

Firstly, this gets the `cvId` for the current window, then gets a list of `figIds` (in this example it is instances), before moving them by 90 and then by (-1 -1). The last step again sees the stretch being performed.

schSubSelectBox

```
schSubSelectBox(
     [ w_windowId ]
     [ g_partial ]
     [ l_bBox ]
     )
     => t / nil
```

Description

Deselects objects within a rectangular area from a specified schematic editor window. Implements the *sub* mode of area selection. With no arguments, it prompts you to enter the deselection area in the current window. Partial deselection is performed if the window environment variable *partialSelect* is set.

Arguments

w_windowId	Window ID of the window containing the objects.
g_partial	Indicates partial selection.
l_bBox	List specifying the corners of the instance box to subselect. If not specified, or specified as `nil`, a bounding box created from all the pins and device shapes is used.

Value Returned

`t`	Deselected objects within a rectangular area from a specified schematic editor window.
`nil`	Deselection was unsuccessful.

Examples

Prompts you to enter the deselection area in the current window of the objects to deselect.

```
schSubSelectBox( )
```

schSymbolToPinList

```
schSymbolToPinList(
     t_libName
     t_cellName
     t_viewName
     )
     => g_pinList / nil
```

Description

Generates a pin list from a symbol cellview.

Arguments

t_libName	Library containing the symbol; must be enclosed in quotation marks.
t_cellName	Cell containing the symbol; must be enclosed in quotation marks.
t_viewName	View containing the symbol; must be enclosed in quotation marks.

Value Returned

g_pinList	Terminal and property information in the form of a pin list, generated from the source symbol.
nil	Pin list generation was unsuccessful.

Examples

Returns the pin list representing the source symbol.

```
pinList = schSymbolToPinList( "myLib" "myDesign" "symbol" )
```

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '( nil ports portList
    [prop proplist] )
portlist = ( termDef termDef...termDef )
termDef = ( nil name "termName"
    direction termDir
    [prop propList]
```

```
    [pins termPins]
    )
proplist = ( nil propName propValue
    propName propValue
    ...
    )
termPins = ( pinDef pinDef...pinDef )
pinDef = ( nil name "pinName"
    [accessDir "accessDir"] )
```

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for usage of schSymbolToPinList in the schViewToPinListReg list of translation functions.

schSync

```
schSync(
     l_cvId
     )
     => t
```

Description

Synchronizes the schematic and Cadence database (CDB) data representations.

Important

You must call this function when you use Cadence database access (CDBA) PI functions, such as `dbCreateInst`, to create a schematic cellview. However, you do not need to call this function when you use PI schematic functions.

Arguments

l_cvId	A list of the schematic cellview IDs (*d_cvId*) whose cellview data representations you want to synchronize with CDB data representations.

Value Returned

Always returns `t`.

Example

Uses `dbCreateInst` to add an instance of `master` to the cellview of `cvId`, and then calls `schSync`, which synchronizes the schematic data representation with the CDB data representation.

```
dbCreateInst( cvId master nil 0:0 "RO" )
schSync( list(cvId) )
```

schTraceNet

```
schTraceNet(
     ?topCellView td_topCellView
     ?net td_net
     [ ?viewList t_viewList ]
     [ ?hierPath lt_hierPath ]
     ?traceCB su_traceCB
     )
     => t / nil
```

Description

Traces a schematic hierarchical net through the design hierarchy and calls a user-defined callback at each level of the hierarchy where the net is present. The user-defined callback is passed as an argument, a user type describing the occurrence of the net being traced at a specific location in the design hierarchy.

 Video

To find out how to use the SKILL `schTraceNet` to calculate the connected area of a net, see Probing and Calculating the Area of a Net.

Arguments

`?topCellView` *td_topCellView*	The design cellview to be searched within. The cellview can be specified by database ID `?topCellView(db:0x1ce0bc9c)` or `?topCellView ("libName" "cellName" "viewName")`. **Note:** `"viewName"` can refer to a Virtuoso config view.
`?net` *td_net*	The net to be traced. Specify the database ID or a string; for example, `"net07"`.
`?viewList` *t_viewList*	A list of view names to be searched within. For example, `"schematic symbol"`.

?hierPath *lt_hierPath*	The hierarchical path to the master cellview, specified either as a string or a list:
	■ string, as returned by `geGetInstHierPath`.
	`"/I2/I0/I1/I0"`
	■ list, as returned by `geGetInstHier`.
	`((db:0x1ce0bc9c 0 0 0)`
	`(db:0x1ce0ba9a 0 0 0)`
	`(db:0x1ce06d1b 0 0 0)`
	`(db:0x1ce0b19a 0 0 0)`
	`)`

?traceCB *su_traceCB*	A user-defined callback that is called when a net is found during hierarchy traversal. It is specified using either a symbol representing the user specified callback or a lambda function: `lambda( (netStruct) info("%s\n" netStruct->netName ).` The `netStruct` is a user type with the following content: `pathToNet`, `netName`, `hierPath`, and `net`. All instances are traced, including ignored instances that are not displayed in the Navigator.

Value Returned

`t`	The net was successfully traced.
`nil`	The net trace failed. This can be due to it missing the cellview, missing the net, no hierarchy to traverse.

Examples

In the following example, the callback `TraceNetCB` is predefined to return the name, length, and width of each instance encountered during the trace.

```
defun( TraceNetCB (netStruct)
    let( ( inst (insts makeTable("visited" nil) ) )
        foreach(instTerm netStruct->net~>allInstTerms
```

```
            inst = instTerm~>inst
            unless(insts[inst]
                insts[inst] = t
                when(inst~>w &&inst~>l
                  info("Instance: %s  length: %s width: %s\n" \
                  strcat(netStruct->pathToNet "/" inst~>name) \
                  inst~>w inst~>l)
                )
            )
        )
    )
)
```

The `schTraceNet` call below traces schematic net ‘avss’ through the design hierarchy and calls the callback `TraceNetCB` at each level where the net is found.

```
schTraceNet(?topCellView cv ?net "avss" ?traceCB 'TraceNetCB)
```

It uses the lambda function *`u_traceCB`* which uses the predefined `netStruct` as `lambda( (netStruct) info("%s\n" netStruct->netName )`. This uses SKILL and contains the following fields: *pathToNet*, *netName* (strings) and *hierPath* and *net.*

Results similar to the following are returned:

```
Instance: /I1/m9  length: 2.5 width: 3.5
Instance: /I1/m7  length: 2.5 width: 3.5
Instance: /I1/m5  length: 2.5 width: 3.5
Instance: /I1/m3  length: 2.5 width: 3.5
Instance: /I1/m1  length: 2.5 width: 3.5
```

Related Topics

Probing and Calculating the Area of a Net

geGetInstHierPath

geGetInstHier

schUnregisterFixedMenu

```
schUnregisterFixedMenu(
     t_category
     )
     => t / nil
```

Description

Unregisters the user-registered fixed menu for a specific cellview type and reassigns the default fixed menu.

Arguments

t_category	Cellview type for which the default fixed menu is restored; must be enclosed in quotation marks. Valid Values: `schematic`, `sheetSchematic`, `indexSchematic`, `symbol`

Value Returned

`t`	Unregistered the user-registered fixed menu for specific cellview type and reassigns the default fixed menu.
`nil`	Unregistering was unsuccessful.

Examples

Unregisters the user-registered fixed menu for schematic cellviews and reassigns the system default fixed menu.

```
schUnregisterFixedMenu( "schematic" )
```

schUnregisterPopUpMenu

```
schUnregisterPopUpMenu(
     t_category
     [ t_mode ]
     )
     => t / nil
```

Description

Unregisters the specific category and access mode for object-sensitive menus (OSMs) and reassigns the system default menus.

Arguments

t_category	Category for which the menu is unregistered.; must be enclosed in quotation marks. Valid Values: `instance`, `schPin` `instPin`, `wire`, `label`, `marker`, `schematic`, `schDefault`, `schSelSet`, `symPin`, `shapes`, `symbol`, `symDefault`, `symSelSet`, `schStandard`, `symStandard`
t_mode	Access mode for which the menu handle is unregistered; must be enclosed in quotation marks. If you do not specify the mode, both modes are unregistered. Valid Values: `read`, `edit`

Value Returned

`t`	Unregisters the specific category and access mode for object-sensitive menus (OSMs) and reassigns the system default menus.
`nil`	Unregistering was unsuccessful.

Examples

Unregisters the current OSM and reassigns the system default `instance` OSM. The system default appears when the cursor is over an instance when the current cellview is opened in read mode.

```
schUnregisterPopUpMenu( "instance" "read" )
```

Reassigns the system default for the instance category for both edit and read modes.

```
schUnregisterPopUpMenu( "instance" )
```

schUnregPostCheckTrigger

```
schUnregPostCheckTrigger(
     s_funcName
     )
     => t / nil
```

Description

Unregisters a post check trigger routine.

Arguments

s_funcName	Name of the routine to unregister; must be the symbol for a routine that was registered using `schRegPostCheckTrigger`.

Value Returned

`t`	Unregistered a post check trigger routine.
`nil`	Unregistering was unsuccessful.

Examples

Unregisters the `'checkTrig` routine.

```
schUnregPostCheckTrigger( 'checkTrig ) => t
```

schUnregPreCheckTrigger

```
schUnregPreCheckTrigger(
     s_functionName
     )
     => t / nil
```

Description

Unregisters a pre-check trigger routine.

Arguments

s_functionName	The symbol for the SKILL function that is to be unregistered.

Value Returned

t	Successfully unregistered a pre-check trigger routine.
nil	Unsuccessful.

Examples

Define the triggers:

```
procedure( checkTrig(cv "d")
    info(".. Check Trigger called for %s/%s/%s\n" cv~>libName cv~>cellName
cv~>viewName)
)
procedure(checkOnceTrig(cv "d")
    info(".. Check Once Trigger called for %s/%s/%s\n" cv~>libName
cv~>cellName cv~>viewName)
)
```

Register the triggers:

```
schRegPreCheckTrigger 'checkTrig
schRegPreCheckTrigger 'checkOnceTrig t
```

Unregister the triggers:

```
schUnregPreCheckTrigger 'checkTrig
schUnregPreCheckTrigger 'checkOnceTrig
```

Related Topics

schRegPreCheckTrigger

schUpdateUserSRCErrorAndWarn

```
schUpdateUserSRCErrorAndWarn(
     x_nUserErr
     x_nUserWarn
     )
     => t / nil
```

Description

Increases the number of errors and warnings that are reported on the Schematic Check dialog box that appears after running the schematic rules checker (SRC) commands. This function is useful if you implement additional checks besides those provided by the schematic rules checker.

Arguments

x_nUserErr	Number of additional user-reported errors.
x_nUserWarn	Number of additional user-reported warnings.

Value Returned

`t`	Increased the number of errors and warnings that are reported on the Schematic Check dialog box that appear after running the SRC commands.
`nil`	Unsuccessful.

Example

The `schRegPostCheckTrigger` function registers a new function which will be called after running the schematic rules checker. The procedure which follows creates markers according to your schematic rule checks, updates the local variables, and requests additional error and warning checks be included in the check report. The `schUpdateUserSRCErrorAndWarn` function shows the total number of errors and warnings in a dialog box after running the check.

```
; Register a function which will be called after running the schematic SRC checks.
schRegPostCheckTrigger( 'userSRC )
)

procedure( userSRC(cv nErr nWarn "dxx" )
     printf( "Running userSRC rules checking....\n" )
     nUserErr = 0
```

```
     nUserWarn = 0
     ; Create markers according to your schematic rule checks. Update the local
     ; variables nUserErr and nUserWarn accordingly.
     ; (code omitted)
     ; Request additional error and warning checks to be included in the check
     ; report.
     schUpdateUserSRCErrorAndWarn( nUserErr nUserWarn )
)
```

Increases the number of errors and warnings reported by the schematic rules checker by six and four, respectively.

```
schUpdateUserSRCErrorAndWarn ( 6 4 )
```

schVerilogToPinList

```
schVerilogToPinList(
     t_libName
     t_cellName
     t_viewName
     )
     => g_pinList / nil
```

Description

Generates a pin list from a Verilog HDL cellview.

Arguments

t_libName	Library containing the Verilog HDL cellview; must be enclosed in quotation marks.
t_cellName	Cell containing the Verilog HDL cellview; must be enclosed in quotation marks.
t_viewName	View containing the Verilog HDL cellview; must be enclosed in quotation marks.

Value Returned

g_pinList	Terminal and property information in the form of a pin list, generated from the source Verilog HDL cellview.
nil	Unsuccessful.

Examples

Returns the pin list representing the source Verilog HDL cellview.

```
pinList = schVerilogToPinList( "myLib" "myDesign" "symbol" )
```

The pin list format represents all the terminals and properties and is stored in a disembodied property list with the following format:

```
g_pinList = '(nil ports portList
    [prop proplist] )
portlist = (termDef termDef...termDef)
termDef = (nil name "termName"
direction termDir
    [prop propList]
```

```
    [pins termPins]
    )
proplist = (nil propName propValue
    propName propValue
    ...
    )
termPins = (pinDef pinDef...pinDef)
pinDef =  (nil name "pinName"
    [accessDir "accessDir"])
```

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for usage of schVerilogIToPinList in the schViewToPinListReg list of translation functions.

schVIC

```
schVIC(
     d_cvId
     )
     => l_result
```

Description

Runs the cross-view checker to check the consistency of the interface of one or more views against the view of the given cellview.

The member terminals of the specified cellview are compared against member terminals of the views specified in the schematic environment `vicViewList` variable. This check flags differences between signals exported from the specified cellview and signals exported in other views of the same cell.

The following types of errors are reported:

- Signals that are exported in one view but not the other
- Signals that have terminals with different directions in the two views

You can use this function to check the consistency between a schematic and its corresponding symbol. All markers generated by this check are indicated with the source of `VIC` and are placed in the given cellview.

The `vicViewList` default in the schematic environment is `symbol`. You can change this with a call to `schSetEnv`. It is not an error if a named view does not exist for the cell.

The current cellview and the views to check must be Cadence databases.

Arguments

d_cvId	Cellview ID of the cellview in which to check the view interface.

Value Returned

l_result	List containing the number of errors and warnings generated by the check.

Examples

Runs the cross-view checker on the cellview `cv`.

```
result = schVIC( cv )
numErrors = car( result )
numWarns = cadr( result )
```

schViewToView

```
schViewToView(
     t_sourceLibName
     t_sourceCellName
     t_libName
     t_cellName
     t_viewFrom
     t_viewTo
     t_fromFunc
     t_toFunc
     )
     => t / nil
```

Description

Generates one type of cellview from another.

See the *your_install_dir*/tools/dfII/samples/local/schConfig.il file for schViewMasters list of translation functions and documentation for creating your own translation functions.

Arguments

t_sourceLibName	Name of the library that contains the data to translate; must be enclosed in quotation marks.
t_sourceCellName	Name of the cell that contains the data to translate; must be enclosed in quotation marks.
t_libName	Name of the existing library that will contain the translated cellview; must be enclosed in quotation marks.
t_cellName	Cell name for the translated cellview; must be enclosed in quotation marks.
t_viewFrom	View name to translate from (the source view); must be enclosed in quotation marks.
t_viewTo	View name to translate to (the destination view); must be enclosed in quotation marks.
t_fromFunc	SKILL procedure to translate from the source view to the intermediate pin list format; must be enclosed in quotation marks.

t_toFunc	SKILL procedure to translate from the intermediate pin list format to the destination view defined by *t_libName*, *t_cellName*, and *t_viewTo*; must be enclosed in quotation marks.

Value Returned

t	Generates one type of cellview from another.
nil	Unsuccessful.

Examples

Generates a Verilog HDL shell (myLib myDesign functional) based on the specified symbol (srcLib srcDesign symbol).

```
schViewToView( "srcLib" "srcDesign" "myLib" "myDesign" "symbol" "functional"
"schSymbolToPinList" "schPinListToVerilog" )
```

schZoomFit

```
schZoomFit(
     f_scale1
     f_scale2
     [ w_windowId ]
     )
     => t
```

Description

Performs a zoom-to-fit with the given zoom scale values. If the schematic cellview contains a sheet border, the first zoom scale value is used. The second zoom scale value is used when there is no sheet border.

Arguments

f_scale1	The zoom scale if the schematic cellview contains a sheet border.
f_scale2	The zoom scale if the schematic cellview does not contain a sheet border.
w_windowId	Window where the function runs. If not specified, the current window is used.

Value Returned

Always returns `t`.

Examples

If the schematic cellview contains a sheet border, it is scaled to `1.0`, which represents fitting the cellview bounding box to the size of the current graphic window. If the schematic cellview contains no sheet border, it is scaled `0.9`, which represents fitting the cellview bounding box to 90% of the current graphic window.

```
schZoomFit( 1.0 0.9 )
```

treeSaveHierarchy

```
treeSaveHierarchy(
     n_sessionWindowNum
     t_file
     )
     => t / nil
```

Description

Saves the entire hierarchy including objects that are not visible (filter settings are ignored in the *Navigator*).

Arguments

n_sessionWindowNum	The session window number where the *Navigator* is docked.
t_file	File name path where the content is to be saved.

Value Returned

`t`	Hierarchy saved.
`nil`	Hierarchy not saved.

Examples

Where `1` is the number of the session window where the *Navigator* is located and `myHierarchy.xml` is the file to store the content in the current working directory.

```
treeSaveHierarchy(1 "./myHierarchy.xml")
```

Related Topics

treeSaveScreen

treeSaveScreen

```
treeSaveScreen(
     n_sessionWindowNum
     t_file
     )
     => t / nil
```

Description

Saves the visible hierarchy (filter settings in the *Navigator* are taken into account).

Arguments

n_sessionWindowNum	The session window number where the *Navigator* is docked.
t_file	File name path where the content is to be saved.

Value Returned

t	Visible hierarchy is saved.
nil	Visible hierarchy is not saved.

Examples

Where `1` is the number of the session window where the *Navigator* is located and `myScreen.xml` is the file to store the content in the current working directory.

```
treeSaveScreen(1 "./myScreen.xml")
```

Related Topics

treeSaveHierarchy

tsg

```
tsg(
     library
     inputFile
     [ templateFile ]
     )
     => t
```

Description

The tsg function is used to generate a schematicSymbol type cellview from a tsg (text-to-symbol generator) file. The tsg file, *inputFile*, is a simple text file that provides a textual description of the symbol.

Note: The tsg file has a pre-specified format which is covered in the Text-to-Symbol Generator appendix in the *Virtuoso Schematic Editor User Guide*.

Arguments

library	Specifies the name of the pre-existing library (or a library ddId) where the symbol view is to be created.
inputFile	Specifies the tsg file to be used to create symbol from. This ASCII file contains the symbol description and is the primary input file that controls the symbol to be generated by tsg().
	As mentioned, the tsg file has a pre-specified format which is discussed in the Text-to-Symbol Generator appendix in the *Virtuoso Schematic Editor User Guide.*
templateFile	The tsg template file is a secondary, optional, file that specifies default controls for symbols to be generated by tsg. The *templateFile* uses the same format as the tsg description file.
	If specified, its symbol parameters are used as defaults when creating the symbol unless they are overridden by the corresponding parameters set out in the *inputFile*. Again, you should see Text-to-Symbol Generator for more information.

Value Returned

`t`	Generates the symbol view.
`nil`	Cannot create symbol or `tsg` parsing has failed.

Examples

This will create a symbol view in `testlib`. The `cellName` and `viewName` will be as specified in the `tsgFile`.

```
tsg("testlib" "/hm/gblack/tsgFile")
```

Related Topics

Text-to-Symbol Generator

3

Library Management Commands

The Cadence® SKILL-based symbol and simulation library generator (S/SLG) is a library management program that generates symbol and simulation views. S/SLG generates a symbol in the library and defines the complete characterization of a symbol for use in computer-aed engineering applications.

This chapter describes the library management SKILL functions used with S/SLG, in alphabetical order.

You can use the `alias` mechanism to give a command a shorter name. For example, the command `alias p lmPrintViewProp` makes `p` an alias name for `lmPrintViewProp`. Thus, the command `p( and2 symbol )` is equivalent to the command `lmPrintViewProp( and2 symbol )`. Then, you can use `p` whenever you would have used `lmPrintViewProp`.

lmCheckTerm

```
lmCheckTerm(
     d_cellList
     [ t_filename ]
     )
```

Description

Checks the consistency of terminal name, type, and wth for all views of each specified cell.

If there is a schematic view, terminals in the schematic view are used as the basis of comparison for each cell. Otherwise, terminals in the symbol view are used as the basis of comparison. When neither schematic nor symbol view exists, no terminal checking is performed. After checking, inconsistent terminal names/wths and input/output types are entified and printed.

Arguments

d_cellList	The name of the cells to be checked, in the form `{ cellName … | t }` If *d_cellList* is `t`, all views of the specified cell are checked.
t_filename	Name of a file where the printout is stored. If not specified, output is displayed on the screen; must be enclosed in quotation marks.

Examples

```
lmCheckTerm( (and2 and3) "check.out" )
```

Checks the terminal consistency of all views for the cells `and2` and `and3` and puts output in the file `check.out`.

```
lmCheckTerm( t )
```

Checks the terminal consistency of all views for each cell in the current library.

lmCheckView

```
lmCheckView(
     d_cellList
     t_viewName
     [ t_fileName ]
     t_expression
     )
```

Description

Evaluates a sequence of expressions for each specified cellview. Checks the property value for consistency.

Property values can be included in an expression. The expressions should conform to SKILL syntax. After evaluation, the result is printed for each expression.

Arguments

d_cellList	The name of the views to be checked, in the form `{ cellName … | t }`
t_viewName	The view name to check. Default: `symbol`
t_fileName	Name of a file where the printout is stored. If not specified, output is displayed on the screen.
t_expression	A property used in an expression refers to the property in `viewName`.

Examples

```
fast_process = t
margin = 1ns
lmCheckView( (and2 and3) symbol
    if( fast_process then
        sum = 5ns
        trmax = 4ns
        tfmax = 3ns
    else
        sum = 9ns
        trmax = 6ns
        tfmax = 5ns
    )
lmGetValue(tr) < trmax
lmGetValue(tf) < tfmax
lmGetValue(tr) + lmGetValue(tf) <- sum + margin
)
```

Checks some conditions for the rise and fall time in the `symbol` view of `and2` and `and3`. The property names `tr` and `tf` contain the rise and fall times. The predefined function `lmGetValue` gets the value of the specified property.

```
lmCheckView( t spice
    gammaValue = lmGetValue(gamma)
    lambdaValue = lmGetValue(lambda)
    gammaValue < 0.4
    lamdaValue < 0.04
)
```

Checks the values of the properties `gamma` and `lambda` in the `spice` view for each cell in the current library.

lmCloseLib

```
lmCloseLib
     => t / nil
```

Description

Closes the previously opened working library.

Note: Call `lmOpenLib` before starting any S/SLG command or SKILL function and call `lmCloseLib` when you are done if you are not using `lmLoadData` to load the library command file.

Arguments

None

Example

```
lmCloseLib
```

Closes a working library previously opened by `lmOpenLib`.

lmDefCell

```
lmDefCell( cellName parameters )
     ; input and output declaration, and various parameters
     ; for symbol generation
     [ input( termName termName … termName ) ]
     [ output( termName termName … termName ) ]
     [ io( termName termName … termName ) ]
     [ switch( termName termName … termName ) ]
     [ defsymbol( symbolArguments ) ]
     ; delay parameters and model/element initialization for
     ; a timing analyzer
     [ delayTable( delayArguments )
     [ taModelInit( modelInitializationArguments ) ]
     [ taElmInit( elementInitializationArguments ) ]
     [ timingViolationTable( timingViolationArguments ) ]
     [ <any S/SLG command or SKILL function> ]
     ;
     ; symbol generation takes place (default)
     symbolGen = t
     ;
     ; timing view generation takes place (default)
     taGen = t
     ;
     ; append properties to symbol
     [ lmDefViewProp( viewPropertyArguments ) ]
     ; append properties to terminals of symbol
     [ lmDefTermProp( terminalPropertyArguments ) ]
     )
```

Description

Defines a cell, generates a symbol, and backannotates objects.

You can use any S/SLG command or SKILL function as an argument. If you specify required parameters, the `symbol` view of the cell can be automatically generated. You can specify time delay information for a timing analyzer view. In addition, you can specify properties of the symbol view for simulation and other purposes.

Properties

Each argument specified in `lmDefCell` is sequentially run in addition to default symbol generation and timing view generation. You can add commands (such as `lmDefViewProp` and `lmDefTermProp`) to manipulate symbol properties. You can specify a complete characterization of the symbol in the argument.

Variables

You can set the values of several global Boolean variables to change default values. The `symbolGen` variable controls symbol generation. The default value, `t`, triggers symbol generation as the first action of `lmDefCell`. If you set `symbolGen` to `nil`, no symbol generation takes place.

The `taGen` variable controls timing view generation. The default value, `t`, enables timing view generation if `delayTable` parameters are specified. If you set `taGen` to `nil`, no timing view generation takes place.

Arguments

g_input	Input terminals.
g_output	Output terminals.
g_io	Input/output terminals.
g_switch	Switch terminals.
defsymbol	Symbol generation parameters. Refer to Appendix B, "Text-to-Symbol Generator," for the format of `defsymbol` arguments.
delayTable	Timing delay for the input/output pairs of the cell.
taModelInit	Model for a timing analyzer to initialize.
taElmInit	Element for a timing analyzer to initialize.
timingViolationTable	Illegal timing.

Example

```
lmDefCell( aoi32
     input(A1 A2 A3 B1 B2)
     output(Y)
     lmDefViewProp(
     ; add some properties to the symbol )
)
```

Generates a `symbol` view for the cell `aoi32`.

lmDefTermProp

```
lmDefTermProp(
     [ t_cellName ]
     [ t_viewName ]
     ( g_terminalList t_propertyName t_propertyValue … t_propertyName =
     t_propertyValue )
     ( g_terminalList t_propertyName = t_propertyValue … t_propertyName
     t_propertyValue )
     )
```

Description

Adds properties to the terminals of the specified cellview. To add terminal properties, add specified properties to the original property list and save the new file.

Variables

The `replaceTermProp` global variable controls the property-adding mode. The default value is `nil` which means the mode is appending. If you set `replaceTermProp` to `t`, the mode is replacement. `replaceTermProp` is automatically set to `nil` when you run `lmDefCell` and reset to the original value when finished.

Arguments

t_cellName	Optional only when `lmDefTermProp` is used in `lmDefCell`. The first name (not a terminal name) in the argument list is treated as the cell name.
t_viewName	Default: `symbol`
g_terminalList	Has the following format: `[ terminalName | "t" | "input" | "output" | "io" | "switch" | ( terminalName … ) ]` where `t` implies all terminals of the cell, `input` implies all input terminals, `output` implies all output terminals, `io` implies all input/output terminals, and `switch` implies all switch terminals.
t_propertyName	Any SKILL symbol (entifier) or string.
t_propertyValue	Any SKILL expression that returns proper values.

Example

```
lmDefTermProp( and2 symbol
    ( A name = "A" )
    ( B name = "B" )
    ( Y name = "Y" )
)
lmDefTermProp( and2
    ( t create = time("Apr 15 9:00:00 2000") )
    ; refer to all terminals
)
lmDefTermProp( and2 symbol
    ( input a = 1 )
    ; refer to all input terminals
    ( Y b = 1.1)
)
lmDefTermProp( and2
    ( (A B) type = "input")
)
```

Cell `and2` has input terminals `A` and `B` and output terminal `Y`. Puts a list of properties into these terminals.

lmDefViewProp

```
lmDefViewProp(
     [ t_cellName ]
     [ t_viewName ]
     ( t_propertyName t_propertyValue … t_propertyName t_propertyValue )
     )
```

Description

Adds properties to the specified cellview. To add properties, add the specified properties to the original property list and save the new file.

Variables

The `replaceViewProp` global variable controls the property adding mode. The default value, `nil`, means the mode is appending. If you set `replaceViewProp` to `t`, the mode is replacement. `replaceViewProp` is automatically set to `nil` when running `lmDefCell` and is reset to the original value when finished.

Arguments

t_cellName
Optional only when `lmDefViewProp` is used in `lmDefCell`. The first name (not a property name) in the argument list is treated as the cell name.

t_viewName
Default: `symbol`

t_propertyName
Any SKILL symbol (entifier) or string.

t_propertyValue
Any SKILL expression that returns proper values. The syntax is as follows:

```
propertyValue := simpleValue
( simpleValue [( range )]) |
time( TimeValue [(range)]) |
filename( filenameValue ) |
ilExpr( string ) |
nlpExpr( string ) |
proplist( propertyList )
    propertyList := propertyPair propertyList |
    propertyPair propertyPair := propertyName =
    propertyValue
range := enumeration | lowerBound upperBound
enumeration := stringEnumeration | string
lowerBound := simpleValue | TimeValue | nil
upperBound := simpleValue | TimeValue | nil
simpleValue := integer | floatingPointNumber | string
BooleanValue := yes | true | t | no | false | nil
TimeValue := string
filenameValue := string
```

time
Indicates the property type is time. If the type of a property is `time`, `TimeValue` should contain the date, time, and year. For example, the property

```
lastChanged = time("Jan 20 8:17:56 2000")
```

is a valid expression. Only the first three characters are needed to specify the month.

filename
Indicates the property type is a filename. If the property type is `filename`, `filenameValue` should be a string indicating the filename. For example, the property

```
myFile = filename("magic.c")
```

is a valid expression.

ilExpr	Indicates the property type is a SKILL expression. This type of property should have a valid SKILL expression.
nlpExpr	Indicates the property type is a Netlist Processor (NLP) expression. This type of property is used mainly by the NLP in generating a netlist. Refer to the *Open Simulation System (OSS) Reference Manual* for details about NLP.
proplist	Indicates the property type is a property list. In the database, a property list can be specified as the value of a property. That is, a property list can contain other property lists to form a hierarchical property list.
range	The values of `range` must be enclosed in parentheses. If `nil` is specified for a lower or upper bound in a range, it is unbounded. If a range is an enumeration type, each value in the range should be a string (a name in quotation marks).

Example

```
lmDefViewProp( and2 symbol
    "instance#" = 1
    snapSpacing = 4
    screenGrSpacing = 20
    screenGrMultiple = 5
    drawGr? = yes
    drawAxes? = no
    userUnits = "userUnits"
    "graphicsEditorUnits per userUnit" = 1
    newFont2 = time("Mar 19 14:38:57 2000")
    lastChecked = time("Nov 27 14:30:57 1999")
    tr = 2.0 * basic_process_time
    tf = 0.5 * lmGetValue(tr)
    ; properties for simulation (a hierarchical propertylist
    logic = proplist(
        Input_Pin_List = nlpExpr("[|A] [|B]")
        Pin_Net_Map = nlpExpr("\\n$ 1 [|Y]=Y [|A]=A [|B]=B")
            ; the value of Pin_Net_map is nlpExpr("\n$ 1
            ; [|Y]=Y [|A]=A [|B]=b")
            ; the first "\" is used to quote the second for parsing
        NLPElementPostamble = nlpExpr("[@logic_AND_Image]")
    )
    ; property with name="speed", value=10, type=integer,
    ; lower-bound=8 and upper-bound=12
    speed = (10 (8 12))
    ; property with name="file", value="property.c", type=filename
```

```
    file = filename("property.c")
)
```

Puts a list of properties into the `symbol` view of the cell `and2`. Before you run this function, you should define the variable `basic_process_time`. The `lmGetValue` function is a predefined function for getting the value of a named property.

```
ten = 10                          ; define variable ten
unit = 1n                         ; define variable unit
lmDefViewProp( and2 "cmos_sch"    ; cmos_sch is double quoted because
                                  ; the non-alphanumeric character
                                  ; "." is in the name
    a = 10
    b = (ten (8 12))
    c = (10)
    d = (ten (8 nil))
    e = 2*unit
    f = (2n (1n 4n))
    g = (2n)
    h = (2*unit (nil 4*unit))
    i = yes                       ; yes, true, and t are entical
    j = true
    k = t
    l = no                        ; no, false, and nil are entical
    m = false
    n = nil
    o = yes
    p = "test"
    q = ("test" ("try" "test" "experiment"))
    r = ("test")
    s = ("test" ("test" "try" "debug"))
    "t" = filename("print.c")     ; name t is double quoted
                                  ; to distinguish it from
                                  ; SKILL keyword t
    u = filename("print.c")       ; do not specify a range
                                  ; for filename
    v = time("Apr 1 12:00:00 2000")
    w = time("Apr 1 12:00:00 2000" ("Mar 1 12:00:00 2000"
        "May 1 12:00:00 2000"))
    x = time("Apr 1 12:00:00 2000")
    y = time("Apr 1 12:00:00 2000" (nil "May 1 12:00:00 2000")
    z = proplist(                 ; hierarchical property list
    aa = 1
    bb = 1.1
    cc = proplist(
        aaa = 2
        bbb = 2.2
        )
    )
)
```

Property specification for the `cmos_sch` view of the cell `and2`.

lmDeleteTermProp

```
lmDeleteTermProp(
     d_cellList
     t_viewName
     ( g_terminalList … t_propertyName g_terminalList … t_propertyName )
     )
```

Description

Deletes specified properties from the terminals of specified views.

Arguments

d_cellList	Specifies the cells of the library, in the form `{ cellName | t | ( cellName … ) }` If *d_cellList* is `t`, all cells in the current library are implied.
t_viewName	The view type. Valid Values: `symbol`, `schematic` Default: `symbol`
g_terminalList	The terminals from which to delete properties, in the form `{ terminalName | t | input | output | io | switch | ( terminalName … ) }` *terminalName* — Name of the terminal: must be enclosed in quotation marks. `t` — Implies all terminals of the cell. `input` — Implies all input terminals. `output` — Implies all output terminals. `io` — Implies all input/output terminals. `switch` — Implies all switch terminals.

Example

```
lmDeleteTermProp( SN74181 symbol
    (input Iih)
    (output Iol Ioh)
)
```

Deletes the property `Iih` from all input terminals and the properties `Iol` and `Ioh` from all output terminals for the `symbol` view of cell `SN74181`.

lmDeleteViewProp

```
lmDeleteViewProp(
     d_cellList
     t_viewName
     t_propertyName … t_propertyName
     )
```

Description

Deletes specified properties from the specified view.

Arguments

d_cellList	Has the following format: `{ cellName | t | ( cellName … ) }`
t_viewName	Default: `symbol`
t_propertyName	Any SKILL symbol (entifier) or string.

Examples

```
lmDeleteViewProp( (and2 and3) symbol partName slot )
```

Deletes the properties `partName` and `slot` from `and2` and `and3` cells `symbol` view.

```
lmDeleteViewProp( t spice gamma lambda )
```

Deletes the `gamma` and `lambda` properties from the `spice` view for each cell in the current library.

lmGetValue

```
lmGetValue(
     t_propertyName
     )
```

Description

Returns the value of a specified view property in the current referenced view.

If the specified property does not exist, `nil` is returned. The returned value can be an integer, a floating-point number, a Boolean value, or a string.

Use `lmGetValue` only in expressions in `lmDefViewProp`, `lmDefTermProp`, or `lmCheckView` to remove any ambiguity about the referenced cellview.

Arguments

t_propertyName	Any SKILL symbol (entifier) or string.

lmLoadData

```
lmLoadData(
     t_file
     t_lib [ { t_path | nil } [ { t_config | nil } [ { t_mode | nil } ] ] ]
     )
     => t / nil
```

Description

Opens a working library, loads the library command file, and closes the library when you are finished.

Arguments

t_file	S/SLG library command filename.
t_lib	Working library name.
t_path	Search path of the working library. If you specify a null string or `nil`, the program searches paths previously set up by `dbSetPath` for the named library.
t_config	Configuration name, which lets you to set up the working context of the library. If you specify a null string or `nil`, the program uses the default configuration.
t_mode	Valid Values: `r` (read only), `a` (append), `w` (write only) Default: `r`

Example

```
lmLoadData( "myFile.lm" "myLib" "" "" "a" )
```

Loads a command file, where `myFile.lm` is the library command file, and `myLib` is your working library.

lmOpenLib

```
lmOpenLib(
     t_lib [ { t_path | nil } [ { t_config | nil } [ { t_mode | nil } ] ] ]
     )
     => t / nil
```

Description

Opens a library.

Note: Call `lmOpenLib` before starting any S/SLG command and call `lmCloseLib` when you are done if you are not using `lmLoadData` to load the library command file.

Arguments

t_lib	Working library name.
t_path	Search path of the working library. If you specify a null string or `nil`, the program searches paths previously set up by `dbSetPath` for the named library.
t_config	Configuration name, which lets you set up the working context of the library. If you specify a null string or `nil`, the program uses the default configuration.
t_mode	Valid Values: `r` (read only), `a` (append), `w` (write only) Default: `r`

Example

```
lmOpenLib( "myLib" )
```

Opens a library in read mode, where `myLib` is your working library.

lmPrintLibTermProp

```
lmPrintLibTermProp(
     t_viewName
     [ t_filename ]
     )
```

Description

Prints terminal properties of the specified view for each cell in the current library.

Arguments

t_viewName	Has the following format: `{ viewName | t | ( viewName … ) }`
t_filename	Name of a file where the printout is stored. If not specified, output is displayed on the screen.

Example

```
lmPrintLibTermProp( symbol )
```

Prints all terminal properties of the `symbol` view for each cell in the current library.

lmPrintLibViewProp

```
lmPrintLibViewProp(
     t_viewName
     [ t_filename]
     )
```

Description

Prints properties of the specified view for each cell in the current library.

Arguments

t_viewName	Default: `symbol`
t_filename	Name of a file where the printout is stored. If not specified, output is displayed on the screen.

Examples

```
lmPrintLibViewProp( symbol )
```

Prints properties of the `symbol` view for each cell in the current library.

```
lmPrintLibViewProp( symbol "symbol.lil" )
```

Stores output in the file `symbol.lil`.

lmPrintTerm

```
lmPrintTerm(
     t_cellName
     t_viewName
     )
```

Description

Prints names, types, and wths of all terminals of the specified cellviews.

Arguments

t_cellName	Optional only when `lmPrintTerm` is used in `lmDefCell`.
t_viewName	View name of the cellview to be printed.

Example

```
lmPrintTerm("nand2" "symbol" )
```

Prints names, types and wths of all terminals of the `nand2 symbol` cellview.

lmPrintTermProp

```
lmPrintTermProp(
     t_cellName
     l_viewList
     [ t_filename ]
     )
```

Description

Prints properties of all terminals of the specified views of a cell.

Arguments

t_cellName	Optional only when `lmPrintTermProp` is used in `lmDefCell`.
l_viewList	Has the following format: `{ viewName | t | ( viewName … ) }`
t_filename	Name of a file where the printout is stored. If not specified, output is displayed on the screen.

Output from `lmPrintTermProp` has the same format as that of `lmDefTermProp`, so you can edit the output file, and then put the properties back into the terminals.

Example

```
lmPrintTermProp( nfet t )
```

Prints the terminal properties of all views of the cell `nfet`.

lmPrintViewProp

```
lmPrintViewProp(
     t_cellName
     l_viewList
     [ t_filename ]
     )
```

Description

Prints properties of all specified views of a cell.

Arguments

t_cellName	Optional only when `lmPrintViewProp` is used in `lmDefCell`. The first name (not a property name) in the argument list is treated as the cell name.
l_viewList	Has the following format: `{ viewName | t | ( viewName … ) }`
t_filename	Name of a file where the printout is stored. If not specified, the output is displayed on the screen.

Output from `lmPrintViewProp` has the same format as that of `lmDefViewProp`, so you can edit the output file and then put the properties back into the view.

Examples

```
lmPrintViewProp( and2 symbol "and2.out" )
```

Prints properties of the `symbol` view of the cell `and2` in the file `and2.out`.

```
lmPrintViewProp( and2 t "and2prop" )
```

Prints properties of all views of the cell `and2` in the file `and2prop`.

```
lmPrintViewProp( and2 (symbol ta) "and2.sim" )
```

Prints properties of views `symbol` and `ta` of the cell `and2` in the file `and2.sim`.

lmReset

```
lmReset( )
```

Description

Resets all global variables to their default values.

Arguments

None

lmSimView

```
lmSimView(
     t_cellName
     t_templateViewName
     t_targetViewName
     [ t_targetViewtype ]
     t_propertyName = t_propertyValue … t_propertyName t_propertyValue
     )
```

Description

Creates a view with specified properties for the specified cell.

Arguments

t_cellName	Optional only when `lmSimView` is used in `lmDefCell`.
t_templateViewName	Name of a template view, normally symbol, in which the terminal information is stored.
t_targetViewName	Name of the view to be created.
t_targetViewtype	Type of the view to be created. Use this optional argument if you do not want to overwrite the current view type. The view type `schematicSymbol` is the default.
t_propertyName	Any SKILL symbol (entifier) or string.
t_propertyValue	Any SKILL expression, as long as the expression returns proper values.

All specified properties are added into the target view. The arguments `cellName` and `templateViewName` must exist in the current library.

Example

```
lmSimView( and2 symbol
    CreationTime = time("May 1 10:00:00 2000")
    Input_Pin_List = nlpExpr("[|A] [|B]")
    NLPElementPostamble = nlpExpr("[|Y] .AND
    [@Input_Pin_List]
    [@NLPElement]")
)
```

simRep

```
simRep(
     t_cellName
     t_templateViewName
     t_targetViewName
     [ t_targetViewtype ]
     t_propertyName = t_propertyValue … t_propertyName t_propertyValue
     )
```

Description

Creates a view with specified properties for the specified cell. Use this function only for backward compatibility with Edge 2.1; otherwise, use the `lmSimView` function.

Arguments

t_cellName	Optional only when `lmSimView` is used in `lmDefCell`.
t_templateViewName	Name of a template view, normally symbol, in which the terminal information is stored.
t_targetViewName	Name of the view to be created.
t_targetViewtype	Type of the view to be created. Use this optional argument if you do not want to overwrite the current view type. The view type `schematicSymbol` is the default.
t_propertyName	Any SKILL symbol (entifier) or string.
t_t_propertyValue	Any SKILL expression, as long as the expression returns proper values.

All specified properties are added into the target view. The arguments `cellName` and `templateViewName` must exist in the current library.

Example

```
SimRep( and2 symbol
    CreationTime = time("May 1 10:00:00 2000")
    Input_Pin_List = nlpExpr("[|A] [|B]")
    NLPElementPostamble = nlpExpr("[|Y] .AND
    [@Input_Pin_List]
    [@NLPElement]")
)
```

4

Schematic Assistant Commands

schAsstAddSelectBox

```
schAsstAddSelectBox(
     )
     => Error message
```

Description

Only to be used as the EnterFunction Command string of a bindkey and should not be called otherwise.

Arguments

None

Value Returned

Error message	Returns an error message that states this command only exists for bindkey customization in the Schematic assistant.

Examples

Registers a mouse bindkey.

```
hiSetBindKey("Schematic Assistant" "Shift<DrawThru1>" "schAsstAddSelectBox()")
```

schAsstAutoZoomPan

```
schAsstAutoZoomPan(
     [ w_windowId ]
     [ t_mode ]
     [ x_scale ]
     )
     => t / nil
```

Description

Performs auto-zoom on or pans to the selected objects.

Arguments

w_windowId	Window containing the Schematic assistant.
t_mode	Possible values are: ■ Pan To Selected ■ Zoom To Selected ■ Zoom To Fit
x_scale	Specifies the zoom to fit scale. The default is equal to the value of the autoZoomScale environment variable. The default is `90`.

Value Returned

`t`	Object is selected.
`nil`	Object was not selected.

Examples

Pans to the selected set:

```
schAsstAutoZoomPan(hiGetCurrentWindow() "Pan To Selected")
```

Zooms to the selected set:

```
schAsstAutoZoomPan(hiGetCurrentWindow() "Zoom To Selected")
```

schAsstCellViewProperty

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

Description

Raises the Schematic Cell View Properties form the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Raises the Schematic Cell View Properties from the Schematic assistant.

```
schAsstCellViewProperty( )
```

schAsstDescend

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

Description

Traverses down the hierarchy and displays the child cellview of a specified instance and view you select if you have edit permission. If you do not have edit permission, a dialog box prompts you to use read mode.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Performs a descend read on the selected instance.

```
schAsstDescend( )
```

schAsstDisplayMenu

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

Description

Raises the context menu at the cursor location for the Schematic assistant. This allows accessing most of the other commands available in the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Raises the Context Menu at the cursor location for the Schematic assistant.

```
schAsstDisplayMenu()
```

schAsstDynamicPan

```
schAsstDynamicPan(
     )
     => Error message
```

Description

Only to be used as the EnterFunction Command string of a bindkey and should not be called otherwise.

Arguments

None

Value Returned

Error message	Returns an error message that states this command only exists for bindkey customization in the Schematic assistant.

Examples

Registers a mouse bindkey.

```
hiSetBindKey("Schematic Assistant" "<DrawThru2>" "schAsstDynamicPan()")
```

schAsstObjectProperty

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

Description

Raises the Schematic Object Properties form for the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.	Form raised successfully.

Examples

Raises the Schematic Object Properties form for the Schematic assistant.

```
schAsstCellViewProperty( )
```

schAsstProbeSelected

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

Description

Probes the net of the currently selected figure.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Probes the net of the figure currently selected on the Schematic assistant canvas.

```
schAsstProbeSelected()
```

schAsstRemoveAllProbes

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

Description

Deletes all probes in the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Deletes active probes in the Schematic assistant.

```
; select wire
schAsstProbeSelected()
; select pin
schAsstProbeSelected()

schAsstRemoveAllProbes()
```

schAsstRemoveProbe

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

Description

Deletes the probes of the selected figures in the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Deletes all probes active on the selected figures in the Schematic assistant.

```
; select wire
schAsstProbeSelected()
; select pin
schAsstProbeSelected()

; select only the wire
schAsstRemoveProbe()
```

schAsstReturn

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

Description

Returns up the hierarchy. You can use this when editing schematics or symbols after completing a descend action.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Performs a return on the Schematic assistant.

```
schAsstReturn()
```

schAsstReturnToTop

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

Description

Returns to the top-level cellview in the hierarchy. You can use this when editing schematics or symbols after completing a series of descend commands.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Performs a return to the top on the Schematic assistant.

```
schAsstReturnToTop()
```

schAsstScroll

```
schAsstScroll(
     ?dir t_dir
     [ w_windowId ]
     )
     => t
```

Description

Pans the canvas in the Schematic assistant in the passed direction.

Arguments

`?dir` *t_dir*	The possible values are as follows: ■ `n` (North - up) ■ `e` (East - right) ■ `s` (South - down) ■ `w` (West - left) The default is `nil`.
w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Pans the canvas up.

```
schAsstScroll(?dir "n")
```

Pans the canvas right.

```
schAsstScroll(?dir "e")
```

schAsstSelectPT

```
schAsstSelectPT(
     [ w_windowId ]
     [ s_type ]
     )
     => t / nil
```

Description

Selects an object under the mouse cursor in the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.
s_type	Symbol denoting the selection type. Possible values are `single`, `add`, and `sub`. The default is `single`.

Value Returned

`t`	Object below the cursor is selected successfully.
`nil`	There is no valid object under the mouse cursor.

Examples

Selects an object.

```
hiSetBindKey("Schematic Assistant" "<Btn1Down>" "schAsstSelectPt(?type 'single)")
```

Adds an object to the selection.

```
hiSetBindKey("Schematic Assistant" "Shift<Btn1Down>" "schAsstSelectPt(?type
'add)")
```

Removes an object from the selection.

```
hiSetBindKey("Schematic Assistant" "Ctrl<Btn1Down>"  "schAsstSelectPt(?type
'sub)")
```

schAsstSetSelectOptions

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

Description

Raises the Schematic Set Selection Filter form for the Schematic assistant, which gives you more detailed control of the selection filters for the Schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Raises the Schematic Set Selection Filter form.

```
schAsstSetSelectOptions(hiGetCurrentWindow())
```

Related Topics

schSetSelectOptions

schAsstSingleSelectBox

```
schAsstSingleSelectBox(
     )
     => Error message
```

Description

Only to be used as the EnterFunction Command string of a bindkey and should not be called otherwise.

Arguments

None

Value Returned

Error message	Returns an error message that states this command only exists for bindkey customization in the Schematic assistant.

Examples

Registers a mouse bindkey.

```
hiSetBindKey("Schematic Assistant" "<DrawThru1>" "schAsstSingleSelectBox()")
```

schAsstSubSelectBox

```
schAsstSubSelectBox(
     )
     => Error message
```

Description

Only to be used as the EnterFunction Command string of a bindkey and should not be called otherwise.

Arguments

None

Value Returned

Error message	Returns an error message that states this command only exists for bindkey customization in the Schematic assistant.

Examples

Registers a mouse bindkey.

```
hiSetBindKey("Schematic Assistant" "Ctrl<DrawThru1>" "schAsstSubSelectBox()")
```

schAsstZoomAtMouse

```
schAsstZoomAtMouse(
     ?dir s_dir
     [ w_windowId ]
     )
     => t
```

Description

Zooms in or out at the cursor in canvas of the Schematic assistant.

Arguments

`?dir` *`s_dir`*	Symbol denoting the zoom mode. Possible values are `in` and `out`. The default is `in`.
`w_windowId`	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Zooms in at the cursor.

```
hiSetBindKey("Schematic Assistant" "<Btn4Down>" "schAsstZoomAtMouse(?dir 'in)")
```

Zooms out at the cursor.

```
hiSetBindKey("Schematic Assistant" "<Btn5Down>" "schAsstZoomAtMouse(?dir 'out)")
```

schAsstZoomFit

```
schAsstZoomFit(
     f_factorWithSheet
     f_factorNoSheet
     [ w_windowId ]
     )
     => t
```

Description

Zooms the canvas of the Schematic assistant to display the entire schematic.

Arguments

f_factorWithSheet	Zooms to 1.0 zoom scale if the schematic cellview contains a sheet border.
f_factorNoSheet	Zooms to 0.9 zoom scale if the schematic cellview does not contain a sheet border.
w_windowId	Window containing the Schematic assistant.

Value Returned

Always returns `t`.

Examples

Zooms the canvas of the Schematic assistant to display the entire schematic.

```
schAsstZoomFit()
schAsstZoomFit(?windowId myLayWin)
```

schAsstZoomIn

```
schAsstZoomIn(
     )
     => Error message
```

Description

Only to be used as the EnterFunction Command string of a bindkey and should not be called otherwise.

Arguments

None

Value Returned

Error message	Returns an error message that states this command only exists for bindkey customization in the Schematic assistant.

Examples

Registers a mouse bindkey.

```
hiSetBindKey("Schematic Assistant" "<DrawThru3>" "schAsstZoomIn()")
```

schAsstZoomOut

```
schAsstZoomOut(
     )
     => Error message
```

Description

Only to be used as the EnterFunction Command string of a bindkey and should not be called otherwise.

Arguments

None

Value Returned

Error message	Returns an error message that states this command only exists for bindkey customization in the Schematic assistant.

Examples

Registers a mouse bindkey.

```
hiSetBindKey("Schematic Assistant" "Shift<DrawThru3>" "schAsstZoomOut()")
```

schAsstZoomRelativeScale

```
schAsstZoomRelativeScale(
     [ w_windowId ]
     [ f_scale ]
     )
     => t
```

Description

Zooms in or out of the canvas of the schematic assistant.

Arguments

w_windowId	Window containing the Schematic assistant.
f_scale	Specifies the amount of zoom. Values larger than `1.0` zoom in, values smaller than `1.0` zoom out. The default is `1.0`.

Value Returned

Always returns `t`.

Examples

Zooms in to 1.25 magnification.

```
schAsstZoomRelativeScale(?scale 1.25)
```

Zooms out to 0.8 magnification.

```
schAsstZoomRelativeScale(?scale 0.8)
```

schAsstZoomToSelSet

```
schAsstZoomToSelSet(
     [ w_windowId ]
     [ x_scale ]
     )
     => t
```

Description

Zooms in or out the canvas of the Schematic assistant to display the currently selected objects.

Arguments

w_windowId	Window containing the Schematic assistant.
x_scale	Specifies the scale of the zoom on the viewbox. The default is equal to the value of the autoZoomScale environment variable. The default is `90`.

Value Returned

Always returns `t`.

Examples

Scales the zoom to 90 percent.

```
schAsstZoomToSelSet(hiGetCurrentWindow() 90)
```