cadence®

Virtuoso Schematic Editor SKILL Reference

Product Version ICADVM20.1
October 2020

Printed in the United States of America.

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

Contents

2
Virtuoso Schematic Editor Procedural Interface (PI) Functions

189

Preface

This manual proves information for IC schematic capture and simulation environment developers and designers who want to use Cadence® SKILL language functions instead of menu commands in the schematic capture environments.

This manual assumes that you are familiar with the SKILL language.

Important

SKILL commands that are only applicable to the Virtuoso Schematic Editor XL are tagged "***XL Only***". All other SKILL commands can be used with both the L and XL versions of the schematic editor product.

This preface contains the following topics:

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node and methodologies (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in the ICADVM20.1 advanced nodes and advanced methodologies release.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

The license number required for the Virtuoso® Schematic Editor L is: 95100.

For information on licensing in the Virtuoso design environment, see *Virtuoso Software Licensing and Configuration Guide*.

Related Documentation

What's New and KPNS

- *Virtuoso Schematic Editor What's New*
- *Virtuoso Schematic Editor Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Cadence Application Infrastructure User Guide*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

Virtuoso Tools

- *Virtuoso Schematic Editor User Guide*
- *Virtuoso Schematic Editor SKILL Reference*
- *Virtuoso ADE Assembler User Guide*
- *Virtuoso ADE Explorer User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Spectre Circuit Simulator Reference*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website proves a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rap Adoption Kits

Cadence proves a number of Rap Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on Virtuoso Schematic Editor:

- Virtuoso Schematic Editor
- Virtuoso Analog Design Environment
- Using Virtuoso Constraints Effectively
- Virtuoso Spectre Circuit Simulator
- Spectre Simulations Using Virtuoso ADE
- Virtuoso Electrically-Aware Design with Layout Dependent Effects

Cadence also offers the following training courses on the SKILL programming language, which you can use to customize, extend, and automate your design environment:

- SKILL Language Programming Introduction

- SKILL Language Programming
- Advanced SKILL Language Programming

To explore the full range of training courses proved by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu proves consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (proved you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by proving answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Prove Feedback* box.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

axlGetRunStatus

```
axlGetRunStatus(
    t_sessionName                          <- Required argument
    [ ?optionName t_optionName ]           <- Optional keyword argument
    [ ?historyName t_historyName ]         <- Optional keyword argument
    )
    => l_statusValues                      <- Return value
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName` *`t_optionName`* and `?historyName` *`t_historyName`* are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *l_statusValues*.

Example

```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```

Return value

Value of the session name argument

Question mark and argument name before the value of the keyword argument

Question mark and argument name before the value of the keyword argument

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName` *`t_functionName`* `] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
`|`	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument.
	Used with brackets to indicate that you can specify zero or more arguments.
	Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, `t` is the data type in `t_viewName`. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
`a`	array	array
`A`	amsobject	AMS object
`b`	ddUserType	DDPI object
`B`	ddCatUserType	DDPI category object
`C`	opfcontext	OPF context
`d`	dbobject	Cadence database object (CDBA)
`e`	envobj	environment
`f`	flonum	floating-point number
`F`	opffile	OPF file
`g`	general	any data type
`G`	gdmSpecIlUserType	generic design management (GDM) spec object
`h`	hdbobject	hierarchical database configuration object
`I`	dbgenobject	CDB generator object
`K`	mapiobject	MAPI object
`l`	list	linked list
`L`	tc	Technology file time stamp
`m`	nmpIlUserType	nmpIl user type
`M`	cdsEvalObject	cdsEvalObject
`n`	number	integer or floating-point number
`o`	userType	user-defined type (other)
`p`	port	I/O port
`q`	gdmspecListIlUserType	gdm spec list

Prefix	Internal Name	Data Type
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

1

Virtuoso Schematic Editor Human Interface (HI) Functions

The Virtuoso® Schematic Editor SKILL human interface (HI) functions and arguments, presented in alphabetical order in this chapter, 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. The procedural functions are described in Virtuoso Schematic Editor Procedural Interface (PI) Functions.

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.

Refer to *Virtuoso Schematic Editor User Guide* for more information about the schematic editor forms and options.

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.

The full list of Virtuoso® Schematic Editor SKILL human interface (HI) functions and arguments are as follows:

- annToggleInfoBalloonVisibilityStatus
- annPinCurrentInfoBalloon
- cdsName
- cdsNetExpr
- cdsParam
- cdsTerm
- heHiEditConfig
- heHiSetInstBinding
- heHiShowViewsFound
- heHiUpdate
- hiPrevWinView
- hiNextWinView
- schAddSelectPt
- schDirectEdit
- schExtendSelectPt
- schHiAbout
- schHiAlternateView
- schHiAlign
- schHiCellViewProperty
- schHiChangeEditMode
- schHiCheck
- schHiCheckAndSave
- schHiCheckHier

- schHiRenumberAllSheet
- schHiRenumberInstances
- schHiRenumberSheet
- schHiReplace
- schHiResetInvisibleLabels
- schHiReturn
- schHiReturnToTop
- schHiRotate
- schHiRouteFlightLine
- schHiSaveCellView
- schHiSelectAll
- schHiSelectByProperty
- schHiSetSymbolOrigin
- schHiShowScope
- schHiSnapToGrid
- schHiSolder
- schHiSRC
- schHiStretch
- schHiSymStretch
- schHiTree
- schHiUpdatePinOrder
- schHiVHDLProperty
- schHiUpdatePinsFromView
- schHiVIC
- schHiVICAndSave
- schHiViewToView
- schHiZoomToSelSet

- schHiSetOrigin
- schSetSelectOptions
- schSingleSelectPt

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`	Returns `t` if the annotation balloons are displayed or removed from the schematic canvas.
`nil`	Returns `nil` otherwise.

Example

```
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`	Returns `t` if the annotation balloons on the specified window are pinned.
`nil`	Returns `nil` otherwise.

Example

```
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`.

Example

```
schDirectEdit( 1 )
```

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( 2 )
```

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( 3 )
```

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.

See also Bindkeys and Access Keys in the *Virtuoso Design Environment User Guide*.

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. For details, refer to Aligning.

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.

Example

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

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`.

Caution

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

Arguments

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

Value Returned

Always returns `t`.

Example

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

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

schHiCheck

```
schHiCheck(
     [ ?action t_action ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Displays the options form for modifying the check options.

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

Runs the checks that are set on the form.

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

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

Note: See also 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 ]
     [ @rest t_action 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.

`@rest` *`t_action t_refLibs`*
: List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Displays the form for modifying the check hierarchy options.

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

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 "editOptionsAndRun" )
```

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

schHiCloneSymbol

```
schHiCloneSymbol(
     [ ?libraryName t_libraryName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     [ @rest rest ]
     )
     => 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.

`@rest` *`rest`* List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

schHiComputePinRef

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

`@rest` *rest* List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

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.

schHiCopy

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

`@rest` *`rest`* List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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 t )
```

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

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

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

schHiCreateBlockInst

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

`@rest` *`rest`* List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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 BLOCK2" ?viewName
"SYMBOL_NEG" ?blockSampleName "large" ?instanceName "I1 I2" ?pinNameSeed "PIN" )
```

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.

schHiCreateInst

```
schHiCreateInst(
     [ ?libraryName t_libraryName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     [ ?instanceName t_instanceName ]
     [ ?rows x_rows ]
     [ ?columns x_columns ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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" "A B"
4 3 )
```

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.

schHiCreateInstBox

```
schHiCreateInstBox(
     [ ?autoBox g_autoBox ]
     [ @rest rest ]
     )
     => 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`).
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

schHiCreateKanjiSymbol

```
schHiCreateKanjiSymbol(
     [ ?kanjiFile t_kanjiFile ]
     [ ?libName t_libName ]
     [ ?cellName t_cellName ]
     [ ?viewName t_viewName ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

`t`	The Kanji symbol was created.
`nil`	Caller supplied optional arguments that are not a string.

Example

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

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

schHiCreateMappingSchematic

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

Description

Displays the Create a Mapping Schematic form.

For more information see Creating a Mapping Schematic in the *Virtuoso Schematic Editor User Guide*.

Arguments

N/A

Value Returned

`t`	Mapping schematic form displayed.
`nil`	Command failed.

schHiCreateNetExpression

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

Example

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

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

See also The Syntax of an Inherited Net Expression in the *Virtuoso Schematic L User Guide*.

schHiCreateNoteLabel

```
schHiCreateNoteLabel(
     [ ?text t_text ]
     [ ?justify t_justify ]
     [ ?fontStyle t_fontStyle ]
     [ ?fontHeight n_fontHeight ]
     [ ?type t_type ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

schHiCreateNoteShape

```
schHiCreateNoteShape(
     [ ?shape t_shape ]
     [ ?style t_style ]
     [ ?width n_width ]
     [ @rest rest ]
     )
     => 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. For more details, see the *Cadence Application Infrastructure User Guide*.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

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

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.

See also Adding Patchcords Using the Create Patchcord Form in the *Virtuoso Schematic Editor User Guide*.

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.

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. **Note:** See also The Syntax of an Inherited Net Expression in the *Virtuoso Schematic L User Guide*.

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`.

Example

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

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"
?netExpr "[@power:%:vdd!]" )
```

Adds a net expression to an inout offsheet pin.

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

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> data4 data5" ?direction "input" ?usage
"schematic" ?interpret "full" )
```

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

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

Places an array of schematic input pins.

schHiCreateSheet

```
schHiCreateSheet(
     [ ?number n_number ]
     [ ?size t_size ]
     [ ?type t_type ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

schHiCreateSymbolLabel

```
schHiCreateSymbolLabel(
     [ ?labelChoice t_labelChoice ]
     [ ?text t_text ]
     [ ?justify t_justify ]
     [ ?fontStyle t_fontStyle ]
     [ ?fontHeight n_fontHeight ]
     [ ?type t_type ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

schHiCreateSymbolPin

```
schHiCreateSymbolPin(
     [ ?terminalName t_terminalName ]
     [ ?type t_type ]
     [ ?direction t_direction ]
     [ ?interpret t_interpret ]
     [ ?mode t_mode ]
     [ ?incrCount n_incrCount ]
     [ ?location t_location ]
     [ @rest rest ]
     )
     => 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 `schSymbolPinMaster` 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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( "addr<7:0>" "square" "input" "member" "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( "data1 data2 data3" "square" "input" "full" "single" 0
"left" )
```

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

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

Places an array of symbol input pins.

schHiCreateSymbolShape

```
schHiCreateSymbolShape(
     [ ?shape t_shape ]
     [ ?style t_style ]
     [ ?width x_width ]
     [ ?nonModal g_nonModal ]
     [ @rest rest ]
     )
     => 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`
@rest *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Starts creating another rectangle.

schHiCreateWire

```
schHiCreateWire(
     [ ?width n_width ]
     [ ?drawMode g_drawMode ]
     [ ?routeMethod t_routeMethod ]
     [ ?lockAngle t_lockAngle ]
     [ ?nonModal g_nonModal ]
     [ ?color t_color ]
     [ ?lineStyle t_lineStyle ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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.

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 ]
     [ @rest rest ]
     )
     => 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`.

`@rest` *`rest`*

List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

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

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

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

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

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

schHiCreateWireStubs

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

Description

Starts the Create Wire Stubs and Names command.

For more information see Creating Wire Stubs Names in the *Virtuoso Schematic Editor User Guide*.

Arguments

N/A

Value Returned

`t`	Command started.
`nil`	Command failed.

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).

Arguments

None

Value Returned

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

Example

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).

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 ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Examples

```
schHiDeleteSheet( 4 )
```

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

```
schHiDeleteSheet( 3 6 )
```

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

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. For details, refer to Distributing.

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 aligned.

Example

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

schHiDrawSymbolPin

```
schHiDrawSymbolPin(
     [ ?terminalName t_terminalName ]
     [ ?direction t_direction ]
     [ ?interpret t_interpret ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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( "data2" )
```

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.

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`.

See also 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 ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

Changes the current sheet size to `D` and the border to `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 ]
     [ @rest rest ]
     )
     => 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`.
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

schHiExtractConn

```
schHiExtractConn(
     [ ?action t_action ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Extracts the connectivity from the current schematic.

schHiFind

```
schHiFind(
     [ ?propName t_propName ]
     [ ?condOp t_condOp ]
     [ ?propValue t_propValue ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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.

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`.

schHiFollowPin

```
schHiFollowPin(
     [ ?order t_order ]
     [ @rest rest ]
     )
     => 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`
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

See also schHiComputePinRef.

schHiFontUpdate

```
schHiFontUpdate(
     [?increase x_increase]
     [?amount x_amount]
     [x_increase [x_amount] ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

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

Example

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

Increases font size by two user units.

schHiGotoSheet

```
schHiGotoSheet(
     [ ?sheetNum g_sheetNum ]
     [ @rest rest ]
     )
     => 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`.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

```
schHiGotoSheet( 3 )
```

Views or edits sheet number 3 in the current window.

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 ]
     [ @rest rest ]
     )
     => 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`
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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. For details, refer to Ignoring 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.

Example

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

schHiInstToView

```
schHiInstToView(
     [ ?viewName t_viewName ]
     [ ?dataType t_dataType ]
     [ @rest rest ]
     )
     => 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.
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Generates a schematic view.

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 (see schHiDisplayOptions on page 107). 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)

Category	Description
marker	Markers
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`.

schHiMove

```
schHiMove(
     [ ?useSelSet t_useSelSet ]
     [ @rest rest ]
     )
     => 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`
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

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

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() nil hiGetCurrentWindow())
```

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

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

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

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

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())
```

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

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 ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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`.

See also 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`.

schHiRegisterWireStubs

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

Description

Invokes the Register Net Name per Terminal form.

See also: Registering Default Net Names in the *Virtuoso Schematic Editor User Guide*.

Arguments

None.

Value Returned

Always returns `t`.

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 ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

```
schHiRenumberSheet( 1 2 )
```

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.

schHiReplace

```
schHiReplace(
     [ ?replaceAll g_replaceAll ]
     [ ?propName t_propName ]
     [ ?condOp t_condOp ]
     [ ?propValue t_propValue ]
     [ ?newPropName t_newPropName ]
     [ ?newPropValue t_newPropValue ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Examples

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

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

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

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

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 ]
     [ @rest rest ]
     )
     => 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`
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

schHiRouteFlightLine

```
schHiRouteFlightLine(
     [ ?routeMethod t_routeMethod ]
     [ @rest rest ]
     )
     => 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`
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Routes the flight lines using the full routing algorithm.

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 ]
     [ @rest rest ]
     )
     => 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.

`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

```
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 that have names set to `I1` are added to the selected set.

```
schHiSelectByProperty( "deselect" "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.

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. For details, refer to Snapping to Grid.

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.

Example

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() )
```

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 ]
     [ @rest rest ]
     )
     => 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`

`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

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

schHiStretch

```
schHiStretch(
     [ ?routeMethod t_routeMethod ]
     [ ?useSelSet t_useSelSet ]
     [ @rest rest ]
     )
     => 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`.

Example

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

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

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

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

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

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.

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*.

For more information see Displaying Cellview Listings Using the Print Tree in the *Virtuoso Schematic Editor User Guide*.

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).

Example

```
schHiTree()
```

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`.

See also:

- 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`.

See also the *VHDL Toolbox User Guide.*

schHiUpdatePinsFromView

```
schHiUpdatePinsFromView(
     [ ?viewName t_viewName ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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

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

schHiVIC

```
schHiVIC(
     [ ?viewList t_viewList ]
     [ @rest rest ]
     )
     => 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.
`@rest` *rest*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

Runs the cross-view checker between the currently edited cellview and the views named `symbol` and `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 ]
     [ @rest rest ]
     )
     => 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.
`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

Always returns `t`.

Example

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

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

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`.

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, presented in alphabetical order in this chapter, 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.

The full list of Virtuoso® Schematic Editor SKILL procedural interface (PI) functions and arguments are as follows:

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.

Example

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

when(window
    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.

Example

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.

For more information, see the Text-to-Symbol Generator chapter in the *Virtuoso Schematic Editor User Guide*.

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*)

Note: For definitions of the above, see g_propValue in the Text-to-Symbol Generator chapter in the *Virtuoso Schematic Editor User Guide*.

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.

Note: For information on what label attributes are available, see labelAttr in the Text-to-Symbol Generator chapter in the *Virtuoso Schematic Editor User Guide*.

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.

Example

```
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 )
    )
)
```

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` *`t_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.

Example

```
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.

See also hsmSelect.

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.

Example

```
hsmGetSelectedSet()
```

Will get everything

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

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

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

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

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

Will get extra details on selected objects

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.

Example

```
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.

Example

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.

Example

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.

Example

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.

Example

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)
```

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.

Example

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")
)
```

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.

Example

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")
)
```

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.

Example

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)
```

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 OPC set.
nil	Failed to find the specified set.

Example

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))
```

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.

Example

```
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.

Example

To remove an instance 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
```

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.

Example

```
schAddIgnoreProp( ?name "nlAction" ?prompt "ignore all simulators" ?type "string"
?value "ignore" ?enable t )
```

The `nlAction` property is registered.

schAlign

```
schAlign(
     l_list
     s_justify
     )
     => t / nil
```

Description

Aligns the objects in the specified direction in the schematic or symbol view. For details, refer to Aligning.

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`	Returns `t` if alignment is done successfully
`nil`	Returns `nil` otherwise

Example

```
schAlign( geGetSelSet() 'left )
schAlign( geGetSelSet() 'right )
```

schAttachLibToPackageTech

```
schAttachLibToPackageTech(
     t_libName
     )
     =>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.

Example

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.

Example

```
cvId = dbOpenCellViewByType( "mylib" "top" "schematic" "" 'a )
errs = schCheck( cvId )
```

```
nErrors = car( errs )
nWarns = cadr( errs )
```

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 containing sublists of the ID of the cellview and the number of errors encountered.

`nil`
No errors are found in the hierarchy.

Example

```
cvId = dbOpenCellViewByType( "mylib" "top" "schematic" "" 'a )
schSetEnv( "checkHierSave" t )
schSetEnv( "saveAction" "Save" )
errs = schCheckHier( cvId "schematic cmos_sch" "" )
```

Checks the hierarchy starting at `top schematic` in the library `mylib` where the traversal is controlled by the given view name list. 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.

Example

```
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.

Example

Removes the schematic connectivity from the current cellview.

```
cv = geGetEditCellView()
schClearConn(cv)
```

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`	Unsuccessful.

Example

```
symbolId = dbOpenCellViewByType( "sample" "inv" "symbol" "" 'r )
schCloneSymbol( cvId symbolId 0:0 "R0" )
```

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.

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`	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 ]
     [ @rest rest ]
     )
     => 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`

`@rest` *`rest`*	List of additional arguments that can be passed to the function.

Value Returned

`t`	Created offsheet pin references for multisheet designs.
`nil`	Unsuccessful.

Example

```
schComputePinRef( cellview )
```

Produces cross-references on pins in a schematic using default options.

```
schComputePinRef( cellview "design.xref" ?display "on" ?reportFile nil ?reportDups
"off" ?sortByDir "off" ?reportFile " " )
```

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.

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`	Unsuccessful.

Example

```
objId = schCopy( fig1 cv2 list(10.0:5.0 "R0") )
```

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.

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`	Unsuccessful.

Example

```
symbolId = dbOpenCellViewByType( "sample" "inv" "symbol" "" 'r )
instId = schCreateInst( cvId symbolId "I23" 0:0 "R0" )
```

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.

```
intId = schCreateInst( cvId symbolId "I24<0:1>" 0:1 "R90" )
```

Creates an iterated instance of the same inverter. The instance rotates 90 degrees before being placed.

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`	Unsuccessful.

Example

```
cvId = dbOpenCellViewByType( "sample" "inv" "symbol" "" 'a )
boxId = schCreateInstBox( cvId )
```

Creates an instance box in the `inv` symbol cellview based on the pins and device shapes in the cellview.

```
boxId = schCreateInstBox( cvId list(0:0 2:2) )
```

Creates an instance box with the specified `bBox` coordinates.

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.

Example

```
netExprLabelId = schCreateNetExpression( cv "[@power:%:vdd!]" wireId (0:1.875)
"net1" "lowerLeft" "R0" "fixed" 0.125 )
```

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.

See also The Syntax of an Inherited Net Expression in the *Virtuoso Schematic L User Guide*.

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`	Unsuccessful.

Example

```
labelId = schCreateNoteLabel( cv 0.0:1.875 "any Text String" "lowerLeft" "R0"
"fixed" 0.125 "normalLabel" )
```

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.

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`	Unsuccessful.

Example

```
shapeId = schCreateNoteShape( cv "rectangle" "solid" list(0:0 10:10) )
```

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.

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.
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`	Unsuccessful.

Example

```
inputCVId = dbOpenCellViewByType( "basic" "ipin" "symbol" "" 'r )
pinId = schCreatePin( cvId inputCVId "I1" "input" nil 0:0 "R0" )
```

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.

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`	Unsuccessful.

Example

```
sheetInstId = schCreateSheet( indexId 4 "US_8ths" "Asize" "symbol" )
```

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( cvId 1 "" "" "" )
```

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 2 "US_8ths" "Dsize" "symbol" )
```

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( indexID 4 "US_8th" "" "")
```

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.

schCreateSplitPrimarySymbol

```
schCreateSplitPrimarySymbol(
     d_cellId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) 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`	Returns `t` if 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.

Example

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`	Unsuccessful.

Example

```
labelId = schCreateSymbolLabel( cv 0:1.875 "instance label" "[@instanceName]"
"lowerLeft" "R0" "fixed" 0.125 "NLPLabel" )
```

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.

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.
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`	Unsuccessful.

Example

```
symPinId = schCreateSymbolPin( symbolCV masterCV "A" "input" 0:0 "R0" )
```

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.

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`	Unsuccessful.

Example

```
shapeId = schCreateSymbolShape( cv "rectangle" "solid" list(0:0 1:1) )
```

Creates a solid rectangular shape between points 0:0 and 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.

Example

```
schCreateWire( cv "draw" "full" list(0:0 1:0) 0.0625 0.0625 0.0 )
```

Creates a wire from 0:0 to 1:0.

```
schCreateWire( cv "route" "full" list(0:0 1:20) 0.0625 0.0625 0.05 )
```

Routes a wide wire from 0,0 to 1,20.

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. Default: `0.0625`

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.

Example

```
schCreateWireLabel( cv wireId (0:1.875) "net1" "lowerLeft" "R0" "fixed" 0.1 nil )
```

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.

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`	Unsuccessful.

Example

```
schDelete( fig1 )
```

Deletes `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`	Unsuccessful.

Example

```
schDeleteIndex( cv )
```

Deletes the multisheet index and converts the remaining sheet into an ordinary schematic and converts any offsheet pins to schematic pins.

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`	Unsuccessful.

Example

```
schDeleteSheet( cv 3 )
```

Deletes sheet number 3 from the multisheet index schematic.

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.

Example

```
schDeselectAllFig()
```

Deselects all figures from the cellview in the current window.

schDistribute

```
schDistribute(
     l_list
     s_justify
     )
     => t / nil
```

Description

Arranges objects at equal distance in the specified direction in the schematic or symbol view. For details, refer to Distributing.

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.

Example

```
schDistribute( geGetSelSet() 'vertical )
```

```
schDistribute( geGetSelSet() 'horizontal )
```

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`	Unsuccessful.

Example

```
pinFigId = schDrawSymbolPin( cvId "A" "input" list(0:0 0.0625:0 0.0625:0.0625
0:0.0625) )
```

Creates a pin with terminal `A` with `input` direction using a shape specified by four points.

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`	Unsuccessful.

Example

```
pinList = list( "q" "qbar" "d" "clk" "preset" )
schEditPinOrder( cvId pinList nil )
```

Sets the pin order for the cellview ID to `q`, `qbar`, `d`, `clk`, and `preset`.

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`	Unsuccessful.

Example

```
schEditSheetSize( cv "US_8ths" "Asize" "symbol" )
```

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 "" "" "" )
```

Removes any existing sheet borders.

schExistsEditCap

```
schExistsEditCap(
     { t | nil }
     )
     => t / nil
```

Description

Tests if any licenses that support the schematic editing feature are available for checkout. Does not check out any licenses.

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`	Tested if any licenses that support the schematic editing feature are available for checkout.
`nil`	Unsuccessful.

Example

```
schExistsEditCap(t)
```

Tests if the schematic editing feature is available.

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`	Unsuccessful.

Example

```
schExtendSelSet( hiGetCurrentWindow( ) hiGetCommandPoint( ) )
```

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.

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" on page 305 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.

Example

```
schExtractConn( cv )
```

Extracts connectivity for the cellview you specify.

```
schSetEnv( "runSRC" nil )
schSetEnv( "runVIC" nil )
result = schExtractConn( cv )
```

Extracts connectivity for the cellview you specify but does not run the schematic rules checker or the cross-view checker.

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`	Unsuccessful.

Example

```
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(
     )
     => 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.

Example

```
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
```

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.

Example

```
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.

Example

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`	Unsuccessful.

Example

```
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) )
```

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).

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.

Example

```
schGetCheckGroups()

=> ((ercChecks schCheckGroup@0x1155e788))
```

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. Refer to *Virtuoso Schematic Editor User Guide* schematic environment variable descriptions.

Value Returned

g_value	Current value of the specified variable.

Example

```
result = schGetEnv( "maxLabelOffsetUU" )
```

Returns the value of the maxLabelOffsetUU environment variable.

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.

Example

```
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`	Unsuccessful.

Example

```
cvId = dbOpenCellViewByType( "sample" "flipflop" "schematic" "" 'r nil )
instList = schGetMatchingObjects( cvId "cellName" "==" "nand2" nil )
```

Returns all the `nand2` instances in the `flipflop` schematic.

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.

Example

```
cv = geGetEditCellView( )
pinList = schGetPinOrder(cv)
```

Gets the pin order list for `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.

Example

If there are two registered post-check triggers:

```
_cphPostSchExtractTrigger
_schCICheck
```

Then the call `schGetPostCheckTriggers` will return:

```
("_cphPostSchExtractTrigger"
"_schCICheck")
```

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.

Example

If there are two registered pre-check triggers:

```
_cphPreSchExtractTrigger
_schCICheck
```

Then the call `schGetPreCheckTriggers` will return:

```
("_cphPreSchExtractTrigger"
"_schCICheck")
```

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. See also schSetPropertyDisplay.

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.

Example

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
)
```

schGetShapeStyle

```
schGetShapeStyle(
     d_shape
     )
     => l_style
```

Description

(ICADVM20.1 Only) 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`

Example

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.

See also: schSetSignalTypeIntegrity.

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.

Example

```
schGetSignalTypeIntegrity("tieOff" "scan" "inputOutput") ==> "error"
```

schGetSplitInstances

```
schGetSplitInstances(
     d_instId
     )
     => l_splitInstIds / nil
```

Description

(ICADVM20.1 Only) 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	Unsuccessful.

Example

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

(ICADVM20.1 Only) 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	Unsuccessful.

Example

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

(ICADVM20.1 Only) 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	Unsuccessful.

Example

Returns the split-primary, `i0`.

```
schGetSplitPrimaryInst(i0_s1)
```

schGetSplitPrimaryInstTerm

```
schGetSplitPrimaryInstTerm(
     d_splitInstTermId
     )
     => d_instTerm / nil
```

Description

(ICADVM20.1 Only) 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	Unsuccessful.

Example

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.

See also:

schSetWireColor
schGetWireLineStyle
schSetWireLineStyle
schCreateWire

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.

Example

```
schGetWireColor( car( geGetSelectedSet () ) )
```

schGetWireLineStyle

```
schGetWireLineStyle(
     d_wireId
     )
     => t_lineStyleName / nil
```

Description

Returns the line style used to draw a wire segment.

See also:

schSetWireLineStyle
schGetWireColor
schSetWireColor
schCreateWire

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.

Example

```
schGetWireLineStyle( car( geGetSelectedSet () ) )
```

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`	Unsuccessful.

Example

```
schGlueLabel( labelId figId )
```

Glues the label designated by `labelId` to the figure designated by `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.

These variables are defined in “Customizing Global Environment Variables for Form Fields” in Chapter 13, “Customizing the Virtuoso Schematic Editor,” in *Virtuoso Schematic Editor User Guide*.

Arguments

None.

Value Returned

Always returns `t`.

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`.

Example

```
schHDLReturn( )
```

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.

schIgnore

```
schIgnore(
     d_instId
     g_setIgnore
     )
     => t / nil
```

Description

Ignores the instances in the schematic view. It works as a toggle switch, that is, it is used for adding or removing the ignore properties from the instance. You can view the registered ignore properties on the *Ignore Properties* tab by clicking *Options – Editor.* For details, refer to Ignoring Instances.

Arguments

d_instId	Instance ID to be ignored.
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

Example

Select an instance on a schematic canvas and call one of the functions given below. Instead of selecting an instance, you can also mention the instance ID in the function.

For ignoring an instance,

```
schIgnore(?objectId car(geGetSelSet()) ?setIgnore t)
schIgnore(?objectId instId ?setIgnore t)
```

For recognizing an instance,

```
schIgnore(?objectId car(geGetSelSet()) ?setIgnore nil)
```

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.

Example

```
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 unsuccessful.

Example

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...
```

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`	Unsuccessful.

Example

```
lib = ddGetObj( "myLib" )
schInstallHDL( lib "myDesign" "functional" "myDesign.v" )
```

Creates the HDL cellview `myDesign functional` in the library `myLib`.

```
schInstallHDL( lib "myDesign" "functional" "myDesign.v" t t)
```

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.

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	Unsuccessful.

Example

```
schInstToView( inst "functional" "schSymbolToPinList" "schPinListToVerilog" )
```

Generates a Verilog HDL shell from the specified instance.

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`	Unsuccessful.

Example

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`	Unsuccessful.

Example

```
schIsHDLCapEnabled( t )
```

Validates that a license supporting the schematic editing feature is already checked out.

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`	Returns `t` if the hierarchy checker is currently running.
`nil`	Returns `nil` otherwise.

Example

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."
```

See also

```
schCheck()
schCheckHier()
schCheckHierConfig()
schRegPreCheckTrigger()
schRegPostCheckTrigger()
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`	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`	Unsuccessful.

schIsSplitInst

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

Description

(ICADVM20.1 Only) Tests if a specified instance is a split instance.

Arguments

d_instId	The schematic instance ID.

Value Returned

`t`	Confirmed that the instance is a split instance.
`nil`	Unsuccessful.

Example

`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
```

Note: An instance is a split instance if it's master is a split symbol.

schIsSplitPrimaryInst

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

Description

(ICADVM20.1 Only) 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`	Unsuccessful.

Example

Checks if `i0` is a split-primary instance.

```
schIsSplitPrimarySymbol(i0->master)
=> t
schIsSplitPrimaryInst(i0)
=> t
```

schIsSplitPrimarySymbol

```
schIsSplitPrimarySymbol(
     d_cvId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) 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`	Unsuccessful.

Example

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

(ICADVM20.1 Only) 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`	Unsuccessful.

Example

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`	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

(ICADVM20.1 Only) 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`	Unsuccessful.

Example

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
```

schIsViewCapEnabled

```
schIsViewCapEnabled(
     g_printMesg
     )
     => t
```

Description

Obsolete function. No replacement.

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	Unsuccessful.

Example

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`	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.

Example

```
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.

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`	Unsuccessful.

Example

```
hiSetBindKey( "Schematics" "None<Btn1Down>(2) EF" "schMouseApplyOrFinish()" )
```

Binds the left mouse button double-click action during schematic editor enter function commands to `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	Unsuccessful.

Example

```
fig1 = schMove( fig1 cv2 list(10.0:5.0 "R90" ) )
```

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`.

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!"))
)
```

Example

See *Example* section for 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`	Unsuccessful.

Example

```
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`	Unsuccessful.

Example

```
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`	Unsuccessful.

Example

```
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`	Unsuccessful.

Example

```
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`	Unsuccessful.

Example

```
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`	Unsuccessful.

Example

```
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.

See also Plotting Designs in the *Virtuoso Schematic Editor User Guide*.

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`	Unsuccessful.

Example

```
schPlot( "plotTemplate" window(2))
```

From window 2, generates the plot using the `plotTemplate` template file with backannotated values.

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.

See also *Creating Custom Checks* in the *Virtuoso Schematic Editor User Guide*.

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.

Example

```
(schRegisterCheckGroup
    ?name         'ercChecks
    ?title        "ERC"
    ?description  "Electrical Rule Checks"
    ?enabled      t)
=> schCheckGroup@0xfea8b98
```

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.

See also *Creating Custom Checks* in the *Virtuoso Schematic Editor User Guide*.

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.

Example

```
(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
```

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).

Example

```
(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*. Refer to the *Cadence SKILL Language Reference*.
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`	Unsuccessful.

Examples

```
schRegisterFixedMenu( "schematic" myFixedMenu )
```

Registers `myFixedMenu` as the fixed menu to display when the current cellview is a schematic.

```
schRegisterFixedMenu( "symbol" symFixMenu 'symDisableProc 'symEnableProc )
```

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.

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`	Unsuccessful.

Example

```
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 read mode.

```
schRegisterPopUpMenu( "instance" newInstanceMenu "edit" )
```

Registers `newInstanceMenu` as the OSM to display when the cursor is over an instance when the current cellview is opened in edit mode.

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(@rest args)
  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)
```

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 only called on schematics with connectivity that requires checking.

There are limitations to be aware of when using `schRegPostCheckTrigger`:

- `schRegPostCheckTrigger` will not update the last checked timestamp.

  Note: If adding markers with a customer SKILL program, you can update the last checked timestamp using `dbIsConnCurrent()` or `dbSetConnCurrent()`.

- Customer defined SKILL programs should avoid using any `modify` SKILL functions as they can break connectivities.

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`	Unsuccessful.

Example

Registers `userSRC` as a function to be called after the specified cellview is checked.

```
procedure( userSRC(cv nErr nWarn "dxx" )
    let((isCurrent nUserErr nUserWarn)
        isCurrent = dbIsConnCurrent(cv)
        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 )
        when(isCurrent
        dbSetConnCurrent(cv)
        )
    )
)
schRegPostCheckTrigger( 'userSRC )
```

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.

See also: schUnregPreCheckTrigger.

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`	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" " " )
```

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`	Returns `t` if property is removed successfully.
`nil`	Returns `nil` otherwise.

Example

```
schRemoveIgnoreProp( "nlAction" )
```

Returns `t` if removal is successful. If the property does not exist in the registered ignore property set, the function returns `nil`.

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`	Unsuccessful.

Example

```
schRenumberAllSheet( cv )
```

Resequences the sheets contained in the index schematic cellview.

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.

Example

```
schRenumberInstances( cv )
```

Resequences the instances contained in the cellview.

```
schRenumberInstances( cv "cellview" t "Y-X+" 0 "same master" "sample" "buf"
"symbol" )
```

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.

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`	Unsuccessful.

Examples

```
schRenumberSheet( cv 3 4 )
```

Renumbers sheet 3 as sheet 4. Any succeeding sheets are renumbered.

```
schRenumberSheet( cv 4 3 )
```

Renumbers sheet 4 as sheet 3. If sheet 3 exists, the sheets are swapped.

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`	Unsuccessful.

Example

```
instList = list( instId )
schReplaceProperty( instList "instName" "myInst" )
```

Changes the instance name of specified instance to `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.

Example:

```
myPlotSettingsFile="userHomeDir/currentPlotSettings"
procedure( RememberPlotSettings()
     schSaveCurrentPlotOptions( myPlotSettingsFile )
)
regExitBefore( 'RememberPlotSettings )
```

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.

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.

Example

```
pinList = schSchemToPinList( "myLib" "myDesign" "schematic" )
```

Returns the pin list representing the source 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`.

Example

```
schSelectAllFig( )
```

Selects all objects from the cellview in the current window.

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`	Unsuccessful.

Examples

```
schSelectPoint( hiGetCurrentWindow( ) hiGetCommandPoint( ) t nil 0)
```

Deselects all objects. If the cursor is over an object, the object is selected.

```
schSelectPoint( hiGetCurrentWindow( ) hiGetCommandPoint( ) nil t 0 )
```

If the cursor is over the object, the object is added to the selected set.

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.

Example

```
schSetAndLoadTsgTemplateType( "metric" )
```

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.

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.

Example

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`	Unsuccessful.

Example

```
schSetCmdOption( schCreatePinForm 'direction list( "input" "output" ) 1 "toggle
Direction" )
```

Sets the direction field to toggle between `input` and `output` when the `createPin` command is active and you click the right mouse button.

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

```
result = schSetEnv( "maxLabelOffsetUU" 0.0125 )
```

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.

```
schSetEnv( "vicViewList" "layout symbol" )
```

Sets the value of the `vicViewList` environment variable for the cross-view-checker.

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.

Example

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.

Example

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/or value of an object's attribute, property, or parameter. 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.

Example

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)
)
)
```

schSetShapeStyle

```
schSetShapeStyle(
     d_shape
     [ ?color t_colorName ]
     [ ?lineStyle t_lineStyleName ]
     [ ?stipple t_stippleName ]
     [ ?fillStyle t_fillStyleName ]
     [ ?fillColor t_colorName ]
     )
     => t / nil
```

Description

(ICADVM20.1 Only) 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 returned by the `hiGetCurrentDisplayName` function.

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.

Example

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.

See also: schGetSignalTypeIntegrity.

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.

Example

```
schSetSignalTypeIntegrity("tieOff" "scan" "inputOutput" "error") ==> t
```

schSetSplitPrimaryInst

```
schSetSplitPrimaryInst(
     d_splitInstId
     g_primInstId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) 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.

Example

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

(ICADVM20.1 Only) 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.

Example

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`	Unsuccessful.

Example

```
schSetSymbolOrigin( symbolCV 1:0 )
```

Moves the origin of the cellview specified by `symbolCV` to the point 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.
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`	Unsuccessful.

Examples

```
schSetTextDisplayBBox( tdId nil )
```

Sets the bounding box of the text display object as defined by `tdId`.

```
schSetTextDisplayBBox( tdId instId )
```

Sets the bounding box of the text display object as defined by both `tdId` and `instId`.

schSetWireColor

```
schSetWireColor(
     d_wireId
     t_colorName
     ) => t / nil
```

Description

Sets the color used to draw a wire segment.

See also:

schGetWireColor
schGetWireLineStyle
schSetWireLineStyle
schCreateWire

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.

Example

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")
```

schSetWireLineStyle

```
schSetWireLineStyle(
     d_wireId
     t_lineStyleName
     ) => t / nil
```

Description

Sets the line style used to draw a wire segment.

See also:

schGetWireLineStyle
schGetWireColor
schSetWireColor
schCreateWire

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.

Example

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")
```

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`	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`.

Example

```
schSingleSelectBox( )
```

Prompts you to define the area in the current window for selection with the mouse.

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`	Unsuccessful.

schSnapToGrid

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

Description

Places the object on the grid in a schematic or symbol view. For details, refer to Snapping To Grid.

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.

Example

```
schSnapToGrid( anyObjectFigId )
```

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	Unsuccessful.

Example

```
shapeId = schSolder( cv 2:3 )
```

Places a solder dot at the specified location.

schSplitNumber

```
schSplitNumber(
     )
     => string
```

Description

(ICADVM20.1 Only) 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.

Example

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.

Refer to *Virtuoso Schematic Editor User Guide* for environment variable descriptions.

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.

Example

```
result = schSRC( cvId )
numErrors = car( result )
numWarns = cadr( result )
```

Runs the schematic rules checker on the given cellview and extracts the number of errors and warnings from the result.

```
schSetEnv( "srcUnconnectedWires" "ignored" )
schSetEnv( "srcVerilogSyntax" "error" )
schSetEnv( "srcVHDLSyntax" "ignored" )
result = schSRC( cvId )
```

Sets the severity of the three checks—`srcUnconnectedWires`, `srcVerilogSyntax`, and `srcVHDLSyntax`—and then invokes the schematic rules checker.

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.

Example

Example 1

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

Example 2

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`	Unsuccessful.

Example

```
schSubSelectBox( )
```

Prompts you to enter the deselection area in the current window of the objects to deselect.

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	Unsuccessful.

Example

```
pinList = schSymbolToPinList( "myLib" "myDesign" "symbol" )
```

Returns the pin list representing the source 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

```
dbCreateInst( cvId master nil 0:0 "RO" )
schSync( list(cvId) )
```

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.

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.

Example

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
```

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`	Unsuccessful.

Example

```
schUnregisterFixedMenu( "schematic" )
```

Unregisters the user-registered fixed menu for schematic cellviews and reassigns the system default fixed menu.

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`	Unsuccessful.

Examples

```
schUnregisterPopUpMenu( "instance" "read" )
```

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" )
```

Reassigns the system default for the instance category for both edit and read modes.

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`	Unsuccessful.

Example

```
schUnregPostCheckTrigger( 'checkTrig ) => t
```

Unregisters the `'checkTrig` routine.

schUnregPreCheckTrigger

```
schUnregPreCheckTrigger(
     s_functionName
     [ g_onceOnly ]
     )
     => t / nil
```

Description

Unregisters a pre-check trigger routine.

See also: schRegPreCheckTrigger.

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.

Example

1. 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)
)
```

2. Register the triggers

```
schRegPreCheckTrigger 'checkTrig
schRegPreCheckTrigger 'checkOnceTrig t
```

3. Unregister the triggers

```
schUnregPreCheckTrigger 'checkTrig
schUnregPreCheckTrigger 'checkOnceTrig
```

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

```
; 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 )
)
```

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.

```
schUpdateUserSRCErrorAndWarn ( 6 4 )
```

Increases the number of errors and warnings reported by the schematic rules checker by six and four, respectively.

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.

Example

```
pinList = schVerilogToPinList( "myLib" "myDesign" "symbol" )
```

Returns the pin list representing the source Verilog HDL cellview.

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 schVeriloglToPinList 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.

Example

```
result = schVIC( cv )
numErrors = car( result )
numWarns = cadr( result )
```

Runs the cross-view checker on the cellview `cv`.

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

```
schViewToView( "srcLib" "srcDesign" "myLib" "myDesign" "symbol" "functional"
"schSymbolToPinList" "schPinListToVerilog" )
```

Generates a Verilog HDL shell (`myLib myDesign functional`) based on the specified symbol (`srcLib srcDesign symbol`).

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`.

Example

```
schZoomFit( 1.0 0.9 )
```

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.

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*).

See also treeSaveScreen.

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.

Example

```
treeSaveHierarchy(1 "./myHierarchy.xml")
```

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.

treeSaveScreen

```
treeSaveScreen(
     n_sessionWindowNum
     t_file
     )
     => t / nil
```

Description

Saves the visible hierarchy (filter settings in the *Navigator* are taken into account).

See also treeSaveHierarchy.

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.

Example

```
treeSaveScreen(1 "./myScreen.xml")
```

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.

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.

Example

```
tsg("testlib" "/hm/gblack/tsgFile")
```

This will create a symbol view in `testlib`. The `cellName` and `viewName` will be as specified in the `tsgFile`.

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.

Example

```
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`.

Example

```
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.

Example

```
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.

Example

```
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.

Example

```
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

Function for amsDmv(IC6.1.8 Only)

This chapter describes the public SKILL function provided for the AMS Design and Model Validation feature (amsDmv).

dmvStart

```
dmvStart(
     [arg]
     )
     => t / nil
```

Description

(IC6.1.8 Only) Starts new AMS Design and Model Validation (amsDmv) GUI session from Virtuoso.

Argument

None

Values Returned

None

Examples

Start a new amsDmv GUI session.

```
dmvStart()
```

5

VSDP SKILL Functions (IC6.1.8 Only)

This chapter describes the following SKILL functions:

- sipImportLgaBgaTextSkill
- vsdpiWriteCDF
- vsdpiRunDieExport
- vsdpiGetValueOfDieExportField
- vsdpiSaveXML
- vsdpiSetValueOfDieExportField

sipImportLgaBgaTextSkill

```
sipImportLgaBgaTextSkill(
     myCV
     myDieTextFileName
     [ myPadWidth ]
     [ myDelimiter ]
     )
     => none
```

Description

(IC6.1.8 Only) Imports LGA/BGA text on a blank layout. It creates a layout view if it does not exist.

Arguments

d_myCV	The layout cellview.

s_myDieTextFileName	The name of the LGA/BGA text file.
[s_myPadWidth]	The width of the pad to be created.
[s_myDelimiter]	The delimiter used in the LGA/BGA text file.

Value Returned

`t`	Operation was successful.
`nil`	Operation was unsuccessful.

Example

This is an example of how to use the `sipImportLgaBgaTextSkill` function for importing the LGA/BGA die text file.

Get cellView ID on which LGA/BGA text import need to run.

```
     myCV = dbOpenCellViewByType("<LibName>" "<CellName>" "layout"
"maskLayout" "w")
```

Pass the obtained cellView ID in `sipImportLgaBgaTextSkill` along with `s_pDieTextFileName`, `s_pPadWidth`, and `s_pDelimiter parameter`.

```
     sipImportLgaBgaTextSkill( myCV myDieTextFileName @optional (myPadWidth
40.0) (myDelimiter "Space(\" \" )"))

     myCV:Layout cellview id.

     myDieTextFileName:LGA/BGA text file path.

     myPadWidth: pad width to be used.

     myDelimiter: delimiter used in text file.
```

vsdpiWriteCDF

```
vsdpiWriteCDF(
     t_lib
     ? b_overwrite
     ? b_addinjectedprop
     )
     => none
```

Description

(IC6.1.8 Only) Writes key properties from PTF (Part Table Files) as CDF properties. The DE-HDL parts contain properties specified in PTF files at project or cell-level and these properties impact packaging of parts used in the design.

The packager uses key properties to uniquely identify a part while deciding if it can be assigned to a particular package. To use these parts in the VSDP Implementation flow, the properties need to be made available as CDF properties in the packager.

This function reads the PTF files as specified in the project file assigned to a schematic and converts them to the CDF properties.

Arguments

t_lib	The library name for which CDF properties need to be written.
[t_overwrite]	Flag to control if `libInit.il` file is to be overwritten or appended. This is an optional argument with default value as `nil`.
t_addinjectedprop	Flag to control if injected properties need to be added to CDF properties as well. This is an optional argument with default value as `nil`.

Value Returned

None

Example

This is an example showcasing how to use `vsdpiWriteCDF` to update PTF properties as CDF properties so that cells of that library can be used in VSDP Implementation flow.

```
vsdpiWriteCDF(sip_components t nil)
```

vsdpiRunDieExport

```
vsdpiRunDieExport(
     d_cvid)
     => t / nil
```

Description

(IC6.1.8 Only) Runs Die Export on a given cellview.

Arguments

d_cvid	The cellview ID of the layout view on which Die Export will run. Use `dbOpenCellViewByType` to obtain this ID

Value Returned

`t`	Operation was successful.
`nil`	Operation was unsuccessful.

Example

This is an example of how to use `vsdpiRunDieExport` to run Die Export:

```
mylib = "ic_components"
```

; Library name

```
mycell = "tr_switch"
```

; Cell name

```
myview = "layout"
```

; View name

```
cv = dbOpenCellViewByType(mylib mycell myview)
```

; obtained `cvid` through `dbOpenCellViewByType`

```
vsdpiRunDieExport(cvid)
```

vsdpiGetValueOfDieExportField

```
vsdpiGetValueOfDieExportField(
     d_cvid)
     t_fieldName
     [t_tagName]
     => tagName / nil
```

Description

(IC6.1.8 Only) Returns the value of `tagName` from the `die_config.xml` file as a string.

Arguments

d_cvid	The cellview ID of the layout view on which Die Export needs to be run. `dbOpenCellViewByType` can be used to obtain this ID.
t_fieldName	Name of the field parameter whose tag value needs to be returned by using the `die_config.xml` file. The `fieldName` argument should exists in the `die_config.xml` file.
[t_tagName]	Name of the attribute in the `fieldName` argument, whose value is to be returned. This is an optional argument. The default value of this argument is `value`.

Value Returned

tagName	Operation was successful.
`nil`	Operation was unsuccessful.

Example

This is an example of how to use `vsdpiGetValueOfDieExportField`:

For retrieving the value of Virtuoso Floor plan (`s_lVFPMode`) from the `die_config.xml` file,

```
mylib = "ic_components"
```

; Library name

```
mycell = "tr_switch"
```

; Cell name

```
myview = "layout"
```

; View name

```
cvid = dbOpenCellViewByType(mylib mycell myview)
```

; obtained cvid through `dbOpenCellViewByType`

```
vsdpiGetValueOfDieExportField(cvid "s_lVFPMode" "value")
```

vsdpiSaveXML

```
vsdpiSaveXML(
     d_cvid)
     => t / nil
```

Description

(IC6.1.8 Only) Saves the `die_config.xml` file on the disk.

Arguments

d_cvid	The cellview ID of the layout view on which Die Export needs to be run. `dbOpenCellViewByType` can be used to obtain this ID.

Value Returned

`t`	Operation was successful.
`nil`	Operation was unsuccessful.

Example

; This is an example of how to use `vsdpiSaveXML` to save the `die_config.xml` file on the disk.

```
mylib = "ic_components"
```

; Library name

```
mycell = "tr_switch"
```

; Cell name

```
myview = "layout"
```

; View name

```
cvid = dbOpenCellViewByType(mylib mycell myview)
```

; obtained `cvid` through `dbOpenCellViewByType`

```
vsdpiSaveXML(cvid)
```

vsdpiSetValueOfDieExportField

```
vsdpiSetValueOfDieExportField(
     d_cvid)
     t_fieldName
     t_fieldValue
     [t_tagName]
     => t / nil
```

Description

(IC6.1.8 Only) Sets the field values of the `die_config.xml` file without opening the Die Export form. If the `die_config.xml` file does not exist, this function creates an XML file with default values. This function updates the values only in the memory. To update the XML on the disk, use vsdpiSaveXML.

Arguments

d_cvid	The cellview ID of the layout view on which Die Export needs to be run. `dbOpenCellViewByType` can be used to obtain this ID.
t_fieldName	Name of the field parameter whose tag value needs to be returned by using the `die_config.xml` file. The `fieldName` argument should exists in the `die_config.xml` file.
t_fieldValue	Value to be set.
[t_tagName]	Name of the attribute in the `fieldName` argument, whose value is to be returned. This is an optional argument. The default value of this argument is `value`.

Value Returned

`t`	Operation was successful.
`nil`	Operation was unsuccessful.

Example

This is an example of how to use `vsdpiGetValueOfDieExportField`:

For retrieving the value of Virtuoso Floor plan (`s_lVFPMode`) from the `die_config.xml` file,

```
mylib = "ic_components"
```

; Library name

```
mycell = "tr_switch"
```

; Cell name

```
myview = "layout"
```

; View name

```
cvid = dbOpenCellViewByType(mylib mycell myview)
```

; obtained cvid through `dbOpenCellViewByType`

```
vsdpiSetValueOfDieExportField(cvid "s_lVFPMode" "t" "value")
```