cadence™

Digital Design Netlisting and Simulation SKILL Reference

Product Version ICADVM20.1
October 2020

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

Contents

Preface

This manual describes the SKILL APIs of tools used to netlist and simulate digital designs in the Virtuoso® design environment. It provides information on the SKILL functions that you can use with the following Virtuoso applications:

- Virtuoso Verilog Environment for NC-Verilog Integration
- Virtuoso Verilog Environment for SystemVerilog Integration
- Virtuoso Open Simulation System
- Virtuoso VHDL Toolbox

This manual is aimed at designers of integrated circuits and assumes that you are familiar with:

- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably Virtuoso Layout Suite and Virtuoso Schematic Editor.
- The design and use of parameterized cells.
- Component Description Format (CDF), which lets you create and describe your own components for use with ADE.
- Cadence SKILL™ language.

This preface contains the following topics:

- Typographic and Syntax Conventions
- Identifiers Used to Denote Data Types

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 (for example, ICADVM18.1) releases.

Label	Meaning
(ICADVM18.1 Only)	Features supported only in the ICADVM18.1 advanced nodes and advanced methodologies release.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

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

Related Documentation

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

IC6.1.8 Only

- *Virtuoso Layout Suite L User Guide*
- *Virtuoso Layout Suite XL User Guide*
- *Virtuoso Layout Suite GXL Reference*

ICADVM18.1 Only

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*
- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Concurrent Layout User Guide*
- *Virtuoso Design Planner User Guide*
- *Virtuoso Multi-Patterning Technology User Guide*
- *Virtuoso Placer User Guide*
- *Virtuoso Simulation Driven Interactive Routing User Guide*
- *Virtuoso Width Spacing Patterns User Guide*
- *Virtuoso RF Solution Guide*
- *Virtuoso Electromagnetic Solver Assistant User Guide*

IC6.1.8 and ICADVM18.1

- *Virtuoso Abstract Generator User Guide*
- *Virtuoso Custom Digital Placer User Guide*
- *Virtuoso Design Rule Driven Editing User Guide*
- *Virtuoso Electrically Aware Design Flow Guide*

- *Virtuoso Floorplanner User Guide*
- *Virtuoso Fluid Guard Ring User Guide*
- *Virtuoso Interactive and Assisted Routing User Guide*
- *Virtuoso Layout Suite SKILL Reference*
- *Virtuoso Module Generator User Guide*
- *Virtuoso Parameterized Cell Reference*
- *Virtuoso Pegasus Interactive User Guide*
- *Virtuoso Space-based Router User Guide*
- *Digital Design Netlisting and Simulation SKILL Reference*.

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides 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.

Rapid Adoption Kits

Cadence provides a number of Rapid 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 of interest:

- Virtuoso Schematic Editor
- Virtuoso Analog Design Environment
- Spectre Circuit Simulator
- Spectre Simulations Using Virtuoso ADE
- Virtuoso Simulation for Advanced Nodes
- SKILL Language Programming Introduction
- SKILL Language Programming
- Advanced SKILL Language Programming

To explore the full range of training courses provided 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 provides 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 (provided 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 providing 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 *Provide 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

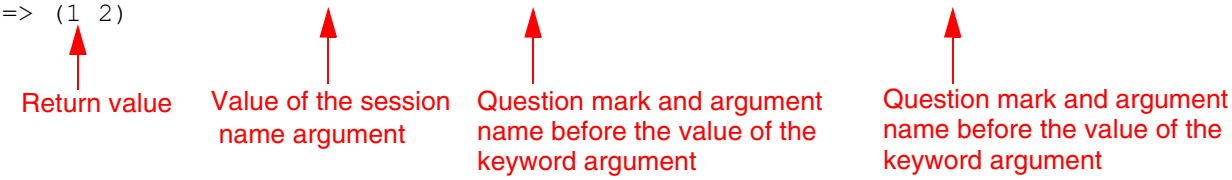
```
axlGetRunStatus(
    t_sessionName                          <-- Required argument
    [ ?optionName t_optionName ]           <-- Optional keyword argument
    [ ?historyName t_historyName ]         <-- Optional keyword argument
    )
    => l_statusValues                      <-- Return value
```

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)
```



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 exactly 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 ID
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

Verilog Netlister Functions

This chapter describes the Verilog netlister functions for invoking the form and update cellviews.

For information on the formatting properties and functions, see *Virtuoso NC-Verilog Environment User Guide*.

vicOpenVlogCallBack

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

Description

The function `vicOpenVlogCallBack()` is a CIW Menu call-back function. This function is called when the Verilog integration environment is invoked. This function invokes the GUI for the Verilog Integration Form.

Arguments

None

Value Returned

`t`	Returns `t` if successful
`nil`	Returns `nil` if unsuccessful.

Example

```
vicOpenVlogCallBack()
```

vlVicCrossSelectionForm

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

Description

This function displays the Cross Selection Setup form. This form is used to cross-select objects of a design between SimVision and Virtuoso Schematic Editor when performing interactive simulation.

Arguments

None

Value Returned

`t`	Returns `t` if successful
`nil`	Returns `nil` if unsuccessful.

Example

```
vlVicCrossSelectionForm()
```

vlVicPSForm

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

Description

This function displays the Post Simulation Analysis form. This form is used to cross-select objects of a design between SimVision and Virtuoso Schematic Editor when performing post-simulation analysis.

Values in the Post Simulation Analysis form can be set using the following SKILL variables:

- *simPSHierPrefix*
- *simPSSimulationDir*
- *simPSSimulationFile*
- *simPSVerilogRunDir*

Arguments

None

Value Returned

`t`	Returns `t` if successful
`nil`	Returns `nil` if unsuccessful.

Example

```
vlVicPSForm()
```

2

OSS Functions

Open Simulation System (OSS) gives you quick access to the simulators that it supports and lets you integrate and customize new simulators into the Virtuoso design environment. For details, see the *Open Simulation System Reference*.

This chapter describes the following SKILL functions associated with OSS:

- FNL Functions
- SE Functions
- SE Graphics Functions
- ISE Functions
- HNL Access Functions
- Miscellaneous Functions
- HNL Trigger Functions for Pre and Post Netlist Customization
- HNL Net-Based Netlisting Functions

FNL Functions

This section describes the flat netlister (FNL) SKILL functions.

fnlAbortNetlist

```
fnlAbortNetlist(
     )
     => nil
```

Description

Aborts netlisting. When the formatter detects an error during netlisting, it calls this function to inform the netlister to abort netlisting.

Arguments

None

Values Returned

`nil`	This function always returns `nil`.

fnlCurrentCell

```
fnlCurrentCell(
     )
     => d_cellviewId / nil
```

Description

Returns the master of the current instance being expanded. The function should be called only during the evaluation of the `NLPcompleteElementString` and `NLPcreateModelString` properties.

Arguments

None

Values Returned

d_cellviewId	Represents an object identifier for the master cell of current instance.
`nil`	Indicates that the function fails to execute successfully.

fnlCurrentCellCdsName

```
fnlCurrentCellCdsName(
     )
     => t_cellName / nil
```

Description

Returns the master cell name of the current instance being expanded. The function should be called only during evaluation of the `NLPcompleteElementString` and `NLPcreateModelString` properties. The function corresponds to the netlister-defined `BlockName` property.

Arguments

None

Values Returned

t_cellName	Represents the name of the master cell of the current instance.
`nil`	Indicates that the function fails to execute successfully.

fnlCurrentInst

```
fnlCurrentInst(
     )
     => d_instanceId / nil
```

Description

Returns the current instance being expanded. The function should be called only during expansion of the `NLPcompleteElementString` property. Do not use the master field of the resulting instanceId. If you use it, you get the symbol cellview rather than the stopping cellview corresponding to the symbol placed in the schematic. To get the instance master in a format instruction, use the `fnlCurrentCell` function.

Arguments

None

Values Returned

d_instanceId	Represents an object identifier for the current instance.
`nil`	Indicates that the function fails to execute successfully.

fnlCurrentInstCdsName

```
fnlCurrentInstCdsName(
     )
     => t_pathName / nil
```

Description

Returns the full instance pathname to the current instance being expanded. The function should be called only during expansion of the `NLPcompleteElementString` property.

Arguments

None

Values Returned

t_pathName	Represents the path to the current instance.
`nil`	Indicates that the function fails to execute successfully.

fnlCurrentIteration

```
fnlCurrentIteration(
     )
     => x_index
```

Description

Returns an index of the current iterated instance being expanded. Only an instance can be iterated and placed in the schematic.

Arguments

None

Values Returned

x_index	Represents the index of the current iterated instance.

fnlCurrentModelExtName

```
fnlCurrentModelExtName(
     )
     => t_modelName / nil
```

Description

Returns the netlister-assigned model name of the current instance being expanded. The function should be called only during evaluation of the `NLPcompleteElementString` and `NLPcreateModelString` properties. The function corresponds to the netlister-defined `ModelNumber` property.

Arguments

None

Values Returned

t_modelName	Represents the netlister-assigned model name of the current instance.
`nil`	Indicates that the function fail has encountered an error while executing.

fnlCurrentSig

```
fnlCurrentSig(
     )
     => d_sigId / nil
```

Description

Returns the current signal being expanded.

Arguments

None

Values Returned

d_sigId	Represents an object identifier for the current signal.
nil	Indicates that the function fail has encountered an error while executing.

fnlCurrentSigPathName

```
fnlCurrentSigPathName(
     )
     => t_pathName / nil
```

Description

Returns the full pathname of the current signal being expanded. It corresponds to the netlister-defined `NetPathName` property. As the corresponding property, this function is valid only during evaluation of the `NLPcreateNetString` property.

Arguments

None

Values Returned

t_pathName	Represents the path of the current signal.
`nil`	Indicates that the function fail has encountered an error while executing.

fnlGetGlobalSigNames

```
fnlGetGlobalSigNames(
     )
     => l_sigNames / nil
```

Description

Returns a list of strings that are the names for all of the global signals contained in the design hierarchy. If the netlister-assigned name is required, you can pass these names to the `fnlSigCdsNameExtName` function to translate them during the header or footer evaluation. The `fnlGetGlobalSigName` function can be called at any time during the netlisting process.

Arguments

None

Values Returned

`l_sigNames`	Represents a list of strings that contain names of the global signals.
`nil`	Indicates that the function fail has encountered an error while executing.

fnlInstCdsNameExtName

```
fnlInstCdsNameExtName(
     t_instName
     )
     => t_netName / nil
```

Description

Returns the netlister-assigned name for the instance name specified as an argument, if the instance is found.

Arguments

t_instName	Represents the instance for which you need to find the netlister assigned name.

Values Returned

t_netName	Represents the netlister-assigned name for the instance.
nil	Indicates that the function fail has encountered an error while executing.

fnlPathList

```
fnlPathList(
     )
     => l_pathList / nil
```

Description

Returns a list representing the current instance path down the schematic hierarchy to the current instance or signal being expanded.

Arguments

None

Values Returned

l_pathList	Represents a list of lists, where each sublist is a (inst cellview index) triplet (the inst member can also be a signal).
nil	Indicates that the function fail has encountered an error while executing.

fnlPrint

```
fnlPrint(
     g_general
     )
     => t / nil
```

Description

Prints its argument to the netlist file.

Arguments

g_general	Represents the information that you need to print to a netlist file. The information can be of any of the SKILL types, such as string, fixnum, or flonum. The argument can also be `t` or `nil`, which do not add any text to the netlist file and are only a convenience.

Values Returned

`t`	Indicates that the function has successfully executed.
`nil`	Indicates that the function fails to execute successfully.

fnlSearchPropString

```
fnlSearchPropString(
     t_propName
     g_localSearch
     )
     => t_propValue / nil
```

Description

Returns property value for the property name specified as an argument.

Arguments

t_propName	Represents the name of the property for which you need to find the value.
g_localSearch	Contains either `nil` or `t`. If *localSearch* is `nil`, the function call corresponds to the substitution expression \"[@ propName]\". If the localSearch argument is `t`, it corresponds to the expression \"[.propName]\".

Values Returned

t_propValue	Represents the property value corresponding to property name specified as an argument.
`nil`	Indicates that the function fail has encountered an error while executing.

fnlSigCdsNameExtName

```
fnlSigCdsNameExtName(
     t_sigName
     )
     => t_netName / nil
```

Description

Returns the netlister-assigned name for the signal name specified as an argument, if the signal is found.

Arguments

t_sigName	Represents the signal name for which you need to find the netlister-assigned name.

Values Returned

t_netName	Represents the netlister-assigned name for the signal.
nil	Indicates that the signal is not found.

fnlTermCdsNameExtName

```
fnlTermCdsNameExtName(
     t_sigName
     )
     => t_netName / nil
```

Description

Returns each netlister-assigned signal name for the signal attached to the terminal whose name is specified as an argument. The function is equivalent to the substitution expression [|name]. As with the matching expression, the terminal names allowed as arguments are restricted to the terminals attached to the current instance.

Arguments

t_sigName	Represents the signal name for which you need to find the netlister-assigned name.

Values Returned

t_netName	Represents the netlister-assigned signal name for the signal specified as an argument.
nil	Indicates that the terminal is not found.

fnlTermExtName

```
fnlTermExtName(
     d_termId
     x_bit
     )
     => t_netName / nil
```

Description

Returns the netlister-assigned name for the signal attached to the bit of the terminal specified as an argument. The function is similar to the substitution expression [|name]. As with the matching expression, the terminals allowed as arguments are restricted to the formal terminals of the master of the current instance. This function should be called only during the evaluation of the `NLPcompleteElementString` property.

Arguments

d_termId	Represents FormalTerminal for which you need to find the netlister-assigned name.
x_bit	Represents the bit of the terminal for which you need to find the netlister assigned name.

Values Returned

t_netName	Represents the netlister-assigned name for the signal returned by the function if the requested bit of the terminal is found.
`nil`	Indicates that the requested bit of the terminal is not found.

fnlTopCell

```
fnlTopCell(
     )
     => d_cellviewId / nil
```

Description

Returns the top level cellview being netlisted. The function can be called throughout the netlist process. It may be especially useful during the evaluation of the NLPnetlistHeader and NLPnetlistFooter properties, where you can use it to output information about the top level design for the header and footer of the netlist. There is no equivalent netlister defined property.

Arguments

None

Values Returned

d_cellviewId	Represents an object identifier of the top-level cellview.
nil	Indicates that the function fail has encountered an error while executing.

SE Functions

The following SKILL functions defined by the simulation environment (SE) let you simplify the integration of your simulator. These functions are in both the Cadence graphics program and the SE program.

cat

```
cat(
     t_filename
     )
     => t / nil
```

Description

Prints the contents of filename on standard output. The function is defined in `etc/skill/si/simcap.ile`.You can modify this function.

Arguments

t_filename	Name of the file.

Value Returned

`t`	Returns `t` if the command was successful.
`nil`	Otherwise, returns `nil`.

Example

```
cat( "si.env" )
```

cdsGetNetlistMode

```
cdsGetNetlistMode(
     )
     => Analog / Digital / Compatibility
```

Description

This function returns a string which is the value of the shell environment variable `CDS_Netlisting_Mode` set by the user in the current shell to invoke Virtuoso.

The valid value for this Unix Shell environment variable are `Digital`, `Analog`, or `Compatibility`. If the user has set any other value, the function would return the value as `Digital`.

Arguments

None

Value Returned

`Digital`	The environment variable `CDS_Netlisting_Mode` set to `Digital`.
`Analog`	The environment variable `CDS_Netlisting_Mode` set to `Analog`.
`Compatibility`	The environment variable `CDS_Netlisting_Mode` set to `Compatibility`.

Example

```
$ virtuoso -nograph
> setShellEnvVar("CDS_Netlisting_Mode=Analog")
t
> cdsSetNetlistMode()
t
> cdsGetNetlistMode()
"Analog"
```

cdsSetNetlistMode

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

Description

This function is used to set the netlisting mode based on the current value of the SHELL environment variable `CDS_Netlisting_Mode`. This SHELL environment variable can have the value *Analog, Digital*, or *Compatibility*. The function `cdsSetNetlistMode` is used from within the Virtuoso SKILL environment to update the settings without restarting Virtuoso.

Arguments

None

Value Returned

`t`	Returns `t` if the command is successful.
`nil`	Otherwise, returns `nil`.

Example

```
$ virtuoso -nograph
> setShellEnvVar("CDS_Netlisting_Mode=Analog")
t
> cdsSetNetlistMode()
t
> cdsGetNetlistMode()
"Analog"
> setShellEnvVar("CDS_Netlisting_Mode=Digital")
t
> cdsSetNetlistMode()
t
> cdsGetNetlistMode()
"Digital"
```

ERC

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

Description

It is the function that defines the sequence of steps for performing an ERC (Electric Rule Checking). If no overriding function has been provided by the user then this function is invoked; it checks the variables the sequence of steps like the netlisting (ercNetlist) and calls the ERC program for execution (ercSimout).

Arguments

None

Value Returned

`t`	Returns `t` on successful completion of ERC.
`nil`	Returns nil if there were any errors during the running of the ERC.

Example

```
ERC()
```

netlist

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

Description

Performs all needed steps to generate a netlist. If the simDoNetlist variable is not t, the function returns t and does not execute.

If the simNetlistHier variable is set, the hnlRunNetlister( ) function is called to generate a hierarchical netlist; otherwise, the values of netlister control variables are printed and the simNetlistWithArgs( ) function is called with the correct arguments to generate a flat netlist. The following variables are passed to `simNetlistWithArgs( )` for use by FNL in producing the netlist:

```
simLibName
simCellName
simViewName
simRunDir
simNlpGlobalLibName
simNlpGlobalCellName
simNlpGlobalViewName
simViewList
simStopList
simGlobalErrFileName
simProbeFileName
simSimulator
simTimeUnit
simCapUnit
simNetlistFileName
```

You must set the above variables correctly before calling this function.

The function is defined in `etc/skill/si/caplib/netlist.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` if the command is successful.
`nil`	Returns `nil` if the command is not successful.

Example

```
netlist()
```

runsim

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

Description

Executes the SKILL instructions stored in the simCommand variable, which is to run the simulator for this simulation. The function then calls `simOutWithArgs` with the correct arguments to translate the textual simulator output stored in the simout.tmp file and produce the si.out file.

The following variables are used as arguments to the simOutWithArgs function and must be correctly set before calling this function.

```
simSedFile
simRunDir
simLibName
simCellName
simViewName
```

For more information on this translation process, refer to the `simOutWithArgs( )` or `simout( )` functions. The return value of the `simCommand` command is returned by this function.

The function is defined in `etc/skill/si/caplib/simulate.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` if command is successful.
`nil`	Returns `nil` if the command is not successful.

Example

```
runsim()
```

sim

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

Description

If this simulation is not being run in batch mode (as determined by the `simBatchFlag` variable being set to `nil`), the `simInitSimulator` function is called. The `simInitSimulator` function is called when SE first starts executing during its initialization phase. The function must be called again if run interactively because the user might have manually set simulator-specific variables such as `silosSimViewList`, which must replace the value of `simViewList` for it to affect netlisting. Calling `simInitSimulator` again ensures correct setting of global variables by simulator-specific variables.

Next, the `simActions` variable is set to the following if it has not already been set:

```
'( simCheckVariables()
   simInitRunDir()
   netlist()
   simin()
   runsim()
)
```

This is the default list of functions, to be executed in order, to run a complete simulation. If these functions do not provide the proper sequence of steps to be performed for a particular simulator, the variable can be set in the simulator-specific file stored in the `local/si/caplib` directory with the same name as the simulator with the `.ile suffix`. The variable can be set inside the file outside of any function or in the function of the same name as the simulator.

Next, each function specified in the `simActions` list is called in order. As soon as one of these functions returns a value other than `t`, the simulation is stopped, the string stored in the `simFailedMessage` variable is printed, and `nil` is returned. If all of the functions return `t`, the string stored in the `simCompleteMessage` variable is printed and `t` is returned. If the `simCompleteMessage` variable or `simFailedMessage` is `nil`, no message is printed.

The function is defined in `etc/skill/si/caplib/simulate.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` on successful completion.
`nil`	Returns `nil` if the command is not successful.

Example

```
sim()
```

simAddProbeCapByName

```
simAddProbeCapByName(
     netName
     )
     => value
```

Description

This function displays the net capacitance of the net name supplied. This function is to be used in conjunction with the function simReadNetCapFile, which reads in the sim.cap file, initializing all the net names in the design with the associated capacitance values. This function will refresh the active display window placing the capacitance value of the net name given.

Arguments

netName	String value to be given to the function which represents the name of the net for which the capacitance value is required.

Value Returned

value	If the net name is valid, the capacitance value will be placed on the net, else capacitance values for all the nets in the current design window will be displayed.

Example

```
simAddProbeCapByName( "net19" )
```

simAddProbeCapByScreen

```
simAddProbeCapByScreen(
     )
```

Description

Displays all the nets capacitance on the current screen.

Arguments

None

Value Returned

None

Example

```
simAddProbeCapByScreen()
```

simAddProbeCapForBusBit

```
simAddProbeCapForBusBit(
     )
```

Description

Displays net capacitance on each bit of a bus.

This functions works the same way as the `simAddProbeCapByName` function except that it will place the capacitance values on each bit of the buses in the design.

Arguments

None

Value Returned

None

Example

```
simAddProbeCapForBusBit()
```

simCheckExist

```
simCheckExist (
     l_variableNames
     )
     => t / nil
```

Description

Returns t if all of the variables in the variableNames list argument are defined and are not set to nil. Otherwise, returns nil and prints an error message.

The function is defined in `etc/skill/si/simcap.ile`.

You can modify this function.

Arguments

l_variableNames	List of variable names.

Value Returned

`t`	Returns `t` if all variables in the list of variable names are defined.
`nil`	Returns `nil` and prints an error message when the command is not successful.

Example

```
simCheckExist('(simSimulator simRunDir) )
```

simCheckHeader

```
simCheckHeader(
     p_inf
     )
     => t / nil
```

Description

Checks the header stamp of the net capacitance file pointer `(sim.cap)`. The header stamp must be the first word on the first line of the `sim.cap` file, and it must be equal to `Net_Capacitance_File`. Use the `infile( )` function to open the `sim.cap` file before calling this function.

Arguments

p_inf	The valid SKILL portId of the input capacitance `sim.cap` file. There is no default value.

Value Returned

`t`	If the file is valid Net Capacitance File.
`nil`	Otherwise, returns `nil`.

Example

```
simCheckHeader( infile("./sim.cap") )
```

simCheckVariables

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

Description

Checks whether the variables simSimulator simCellName, simLibName, simViewName, simRunDir, simViewList, simStopList, simSedFile, simCommand, simNlpGlobalLibName, and simNlpGlobalCellName have been set and are not nil.

It does so by calling the `simCheckExist( )` function.

The function is defined in `etc/skill/si/simcap.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` if all of the variables have been set and are not `nil`.
`nil`	Otherwise, returns `nil` and prints an error message.

Example

```
simCheckVariables()
```

simCheckViewConfig

```
simCheckViewConfig(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Checks if the design you want to netlist is an HDB configuration.

Arguments

t_libName	Name of the library containing the design
t_cellName	Cell name of the design
t_viewName	View name of the design

Value Returned

t	If the given design is an HDB configuration.
nil	If the given design is not an HDB configuration.

Example

```
simCheckViewConfig( t_libName, t_cellName, t_viewName )
```

simDateStamp

```
simDateStamp(
     t_string
     )
     => t / nil
```

Description

Prints the value contained in *t_string* variable followed by the date to standard output.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

t_string	Variable containing a string value.

Value Returned

`t`	Returns `t` if successful.
`nil`	Otherwise, returns `nil`.

Example

```
simDateStamp( "Begin netlisting:" )
```

simDeleteRunDirFile

```
simDeleteRunDirFile(
     t_fileName
     )
     => t / nil
```

Description

Deletes the *t_fileName* file in the simulation run directory.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

t_fileName	Name of the file to be deleted in the run directory.

Value Returned

`t`	Returns `t` if the file deletion is successful.
`nil`	Otherwise, returns `nil`.

Example

```
simDeleteRunDirFile( "raw/waves" )
```

simDesignVarCdsNameExtName

```
simDesignVarCdsNameExtName(
     t_cdsDesignVarName
     )
     => t_string
```

Description

Takes the designer-assigned variable name, `cdsDesignVarName`, as an argument and returns the netlister-assigned name for it as it appears in the netlist. The API reads the mapping information from the simRunDir directory using the design information set through the simCellName, simLibName, simViewName SKILL variables. Therefore, all these variables must be set before calling this API.

Arguments

t_cdsDesignVarName

Name of the design variable as assigned by the designer.

Value Returned

t_string	Netlister-assigned design variable name as it appears in the netlist.

Example

```
simDesignVarCdsNameExtName( "scale" )
returns "_qpar0"
```

simDesignVarExtNameCdsName

```
simDesignVarExtNameCdsName(
     t_extDesignVarName
     )
     => t_string
```

Description

Takes the netlister-assigned variable name, `extDesignVarName`, as an argument and returns the designer-assigned name for it as it appears on the schematic. The API reads the mapping information from the simRunDir directory using the design information set through the simCellName, simLibName, simViewName SKILL variables. Therefore, all these variables must be set before calling this API.

Arguments

t_extDesignVarName

Name of the netlister-assigned design variable name.

Value Returned

t_string	Name of design variable as assigned by the designer, as it appears on the schematic.

Example

```
simDesignVarExtNameCdsName( "_qpar0" )
returns "scale"
```

simDrain

```
simDrain(
     )
     => t
```

Description

This function takes no arguments and executes the equivalent of `drain( stdout )` when executed in the nongraphic environment. No action is taken or required when executed in the Cadence graphics environment.

This is a replacement for the SKILL `drain( )` function. Replace all references to `drain( stdout )` with calls to this function. The function is defined in `bin/si`.You *cannot* modify this function.

Arguments

None

Value Returned

`t`	Always returns `t`.

Example

```
simDrain()
```

simExecute

```
simExecute(
     t_command
     )
     => t / nil
```

Description

Executes the UNIX command or program given as its single argument. The function is defined in `etc/skill/si/simcap.ile`. You can modify this function.

Arguments

t_command	String variable containing the UNIX command.

Value Returned

`t`	Returns `t` if the command is successful and the UNIX exit status is 0.
`nil`	Otherwise, returns `nil`.

Examples

```
simExecute("exec cat si.env")
```

simFindFile

```
simFindFile(
     t_filename
     )
     => t_filename / nil
```

Description

Finds the full file system pathname to *t_filename* in the install hierarchy if it exists. The pathname this function searches for is specified by the `prependInstallPath` function. The function is defined in `bin/si`.You *cannot* modify this function.

Arguments

t_filename	Name of the file to be searched.

Value Returned

t_filename	Returns the full file system pathname of the install hierarchy if it exists.
`nil`	Returns if the pathname cannot be returned.

Example

```
simFindFile("defaults.il")
```

simFlattenWithArgs

```
simFlattenWithArgs(
     t_simLibName
     t_simCellName
     t_simViewName
     t_simRunDir
     t_simNlpGlobalLibName
     t_simNlpGlobalCellName
     t_simNlpGlobalViewName
     l_simViewList
     l_simStopList
     t_simGlobalErrFileName
     t_simProbeFileName
     t_simSimulator
     f_simTimeUnit
     f_simCapUnit
     t_simFlatLibName
     t_simFlatCellName
     t_simFlatViewName
     t_simFlatViewTypeName
     )
     => t / nil
```

Description

For running the `prFlatten` tool. The design hierarchy is specified by `simLibName`, `simCellName`, and `simViewName`. The global formatting properties are defined in `simNlpGlobalLibName`, `simNlpGlobalCellName` and `simNlpGlobalViewName`. The function is defined in `bin/si` and also in the Cadence graphics program.

Arguments

t_simLibName	Name of the library containing the top-level cellview of the design.
t_simCellName	Cell name of the top-level cellview (design) to be netlisted.
t_simViewName	Viewname of the top-level cellview of your design.
t_simRunDir	Full file system pathname to the simulation run directory. This pathname must be the same as the `simRunDir` global SE variable, for example, `"/mnt2/dave/simulations/silos1"`
t_simNlpGlobalLibName	The name of the library that contains the global cellview. The global cellview defines global format strings. Global format strings, in turn, define the netlist syntax. The library name be the same as the `simNlpGlobalLibName` global SE variable, for example, `"basic"`
t_simNlpGlobalCellName	The cell name of the global cellview that contains the global format strings that define the netlist syntax. This cell name must be the same as the `simNlpGlobalCellName` global SE variable, for example, `"nlpglobals"`
t_simNlpGlobalViewName	The view name of the global cellview that contains the global format strings that define the netlist syntax. This view name must be the same as the `simNlpGlobalViewName` global SE variable, for example, `"verilog"`
l_simViewList	View switch list used to determine which view to switch into when a cell has been found. This list must be the same as the `simViewList` global SE variable, for example, `list("verilog" "schematic")`

l_simStopList	Stopping view list used to determine when to halt the expansion of the hierarchy. This list must be the same as the `simStopList` global SE variable, for example, `list("verilog")`
t_simGlobalErrFileName	Name of the file that stores global error messages.
t_simProbeFileName	Name of the probe file that stores errors, usually `probe.err`.
t_simSimulator	Simulator for which the netlist syntax is intended. This simulator name must be the same as the `simSimulator` SE global environment variable, for example, `"verilog"`
f_simTimeUnit	Floating-point time unit factor, for example, 1.0e-9.
f_simCapUnit	Floating-point capacitance unit factor, for example, 1.0e-15.
t_simFlatLibName	Name of the library which contains the flattened cellView.
t_simFlatCellName	Cell name of the flattened cellView.
t_simFlatViewName	View name of the flattened cellView.
t_simFlatViewTypeName	Name of the view type of the flattened cellView.

Value Returned

`t`	Returns `t` if the command is successful.
`nil`	Otherwise, returns `nil`.

Example

```
simFlattenWithArgs( simLibName
   simCellName
   simViewName
   "mnt/dave/chip1/spice1.run"
   simNlpGlobalLibName
   simNlpGlobalCellName
   simNlpGlobalViewName
   simViewList
   simStopList
   "global.err"
   "probe.err"
   simSimulator
```

```
    float(simTimeUnit)
    float(simCapUnit)
    "flat"
    "test"
    "autoLayout"
    "maskLayout"
)
```

simGetLoginName

```
simGetLoginName(
     )
     => t_string
```

Description

Returns the string-valued login name of the current user. The function is defined in `bin/si`. You *cannot* modify this function.

Arguments

None

Value Returned

t_string	Returns the login name of the user.

Example

```
simGetLoginName()
   returns "dave"
```

simGetTermList

```
simGetTermList(
     t_libName
     t_cellName
     t_viewName
     )
     => l_list / nil
```

Description

This function is used to collect all the terminals for a block and return them as a list.

Arguments

t_libName	Name of the library containing the design
t_cellName	Cell name of the design
t_viewName	View name of the design

Value Returned

l_list	SKILL list of the terminals in the specified cellView.
nil	The command is not successful.

Example

```
simGetTermList( "top" "cell" "schematic" )
```

simIlSleep

```
simIlSleep(
     x_seconds
     )
     => t / nil
```

Description

Suspends the current process by the number of seconds specified as an argument. The current process in a replay file launches si in the batch mode and waits for si to return before it executes the next command in the file.

Arguments

x_seconds	Number of seconds by which the execution of a process is to be delayed.

Value Returned

`t`	The command is successfully.
`nil`	The command is not successful.

Example

```
simIlSleep(4)
```

simIfNoProcedure

```
simIfNoProcedure(
     procedureDefinition
     )
     => t
```

Description

Same as the SKILL keyword `procedure`, except the procedure definition given as an argument is only defined if it is not currently defined. This function is used instead of the SKILL `procedure` to permit the overriding of procedures defined in SE by procedures in `.simrc`.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

procedureDefinition	SKILL procedure definition passed as the argument.

Value Returned

`t`	Returns `t` on successful execution of the command.

Example

```
simIfNoProcedure( cat(file)
   let( (cmd status)
      sprintf( cmd "exec cat %s" file )
      status = simExecute(cmd)
      status
   )
)
```

simin

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

Description

Translates the designer-assigned names for nets and instances in the `control` file to the corresponding netlister-assigned names and produces the `si.inp` file for input to the simulator. A message is printed stating that `simin` is being run, and then the `simInWithArgs` function is called with the correct arguments to do the translation.

The following variables are used as arguments to the `simInWithArgs` function and must be correctly set before calling this function:

```
simRunDir
simLibName
simCellName
simViewName
```

All text in the control file is copied to the `si.inp` file unless the text is surrounded by square brackets ([ ]). The opening square bracket ([ ) signals that the following text up to the closing square bracket ( ]) is to be interpreted. The entire expression is replaced by the resulting interpreted value. Following the opening square bracket ([ ) should be one of the following command characters:

`#`	`[#netname]`	Replace the [`#netname`] expression with the netlister-assigned net name for netname.
`$`	`[$instname]`	Replace the [`$instname`] expression with the netlister-assigned instance name for `instname`.
`!`	`[!filename]`	Replace the [`!filename`] expression with the contents of the `filename` file. If you use a relative pathname, the system will look for the file in the simulation run directory. To access a file outside the simulation run directory, use a full file system pathname. If the file does not exist, an error is generated.
`?`	`[?filename]`	Replace the [`?filename`] expression with the contents of the `filename` file. If you use a relative pathname, the system will look for the file in the simulation run directory. To access a file outside of the simulation run directory, use a full file system pathname. If the file does not exist, *no* error is generated.

`n! [n!filename]`	Same as `[! filename]`, except that the contents of the new file are *not* parsed, and square-bracketed expressions are not interpreted.
`n? [n?filename]`	Same as `[? filename]`, except that the contents of the new file are *not* parsed, and square-bracketed expressions are not interpreted.

The function is defined in `etc/skill/si/caplib/siminout.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` on successful completion.
`nil`	Otherwise, returns `nil`.

Example

```
simin()
```

simInitControl

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

Description

If a file called `control` exists in the simulation run directory, this function returns `t`.

If the `simControlFile` variable is `nil`, it locates the files specified by the `simDefaultControl` variable in the `local/si` directory. If not found, it locates the files in the `etc/si` directory using the `prependInstallPath` function and sets `simControlFile` to the resulting full pathname. Next, the function copies the file specified by the `simControlFile` variable to the run directory and names it `control`.

The function is defined in `etc/skill/si/caplib/init.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` if control file is found.
`nil`	Otherwise, returns `nil`.

Example

```
simInitControl()
```

simInitRaw

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

Description

Creates the `raw` directory in the simulation run directory that stores the waveform file. The function is defined in `etc/skill/si/caplib/init.ile`. You can modify this function.

Arguments

None

Value Returned

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

Example

```
simInitRaw()
```

Note: WSF is no longer supported, therefore, this function will be removed in the future release of the product.

simInitRunDir

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

Description

Executes the list of functions specified by the variable *simInitRunActions*. If this variable is not set then `simInitControl` and `simInitRaw` are the default list of functions to be executed in order, which initializes a simulation run directory.

If these functions do not provide the correct sequence of steps to be performed for a particular simulator, the variable can be set in the simulator-specific file.

The variable `simInitRunActions` is set to be

```
'(simInitControl ( )
  simInitRaw( )
)
```

if it has not already been set.

The file is stored in the directory `local/si/caplib` and has the same name as the simulator with the `.ile` suffix, either inside the file outside of any function, or in the function of the same name as the simulator.

Next, each function specified in the list `simInitRunActions` is called in order. As soon as one of these functions returns a value other than `t`, the initialization is stopped, and `nil` is returned. If all the functions return `t`, `t` is returned.

The function is defined in `etc/skill/si/caplib/init.ile`.

Arguments

None

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simInitRunDir()
```

simInitSimulator

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

Description

Calls the function with the same name as the simulator name specified by the `simSimulator` variable (for instance, `silos`). This function must be defined in a file with the same name with either the `.il` or the `.ile` suffix added and stored in the `local/si/caplib` directory.

The function with the same name as the simulator must set the following variables, as well as any simulator-specific variables.

```
simDefaultControl
simViewList
simStopList
simNlpGlobalViewName
simSedFile
simCommand
```

The function is defined in `etc/skill/si/caplib/init.ile`.

You can modify this function.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simInitSimulator()
```

simInstExtNameCdsName

```
simInstExtNameCdsName(
     t_extInstName
     )
     => t_string
```

Description

Takes as an argument the netlister-assigned instance name `extInstName` and returns the designer-assigned instance name as it appears in the schematic.

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

t_extInstName	Netlister-assigned instance name.

Value Returned

t_string	Returns the netlister-assigned instance name `extInstName` given as an argument. The exact name returned depends on the design.

Example

```
simInstExtNameCdsName( "I34" )
returns "/adder1/and1"
```

simInstCdsNameExtName

```
simInstCdsNameExtName(
     t_cdsInstName
     )
     => t_string
```

Description

Takes as an argument the designer-assigned instance name and returns the netlister-assigned instance name for `cdsInstName` as it appears in the netlist. The exact name returned depends on the design.

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

t_cdsInstName	Designer-assigned instance name.

Value Returned

t_string	Returns the netlister-assigned instance name as it appears in the netlist.

Example

```
simInstCdsNameExtName( "/adder1/and1" )
returns "I34"
```

simInWithArgs

```
simInWithArgs(
     l_fileList
     t_runDirName
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Translates the designer-assigned names for nets and instances in the input files into the corresponding netlister-assigned names, and creates the files for input into the simulator.

When the input arguments are processed the following conditions apply:

- If `libName` is `nil`, then the `simLibName`, `simCellName`, `simViewName` environment variables are used.
- Otherwise, if `cellName` is `nil`, then the `simCellName` environment variable is used.
- If `viewName` is `nil`, then the `simViewName` environment variable is used.

Therefore, the `libName` argument must not be `nil` if the `cellName` and `viewName` arguments are used (not `nil`).

All text in the `inputFileName` file is copied to the `outputFileName` file, unless it is surrounded by square brackets ([ ]). The opening square bracket ([) specifies that the following text up to the closing square bracket (]) must be interpreted. The entire expression is replaced by the interpreted value.

One of the command characters below must follow the opening square bracket ([ ):

`#`	`[#netname]`	Replace the [`#netname`] expression with the netlister-assigned node name for `netname`.
`$`	`[$instname]`	Replace the [`$instname`] expression with the netlister-assigned instance name for `instname`.
`!`	`[!filename]`	Replace the `[!filename]` expression with the contents of the filename file. If you use a relative pathname, the system looks for the filename file in the simulation run directory. To access a file outside the simulation run directory, use a full file system pathname. If the filename file does not exist, an error is generated.

`? [?filename]`	Replace the [`?filename`] expression with the contents of the `filename` file. If you use a relative pathname, the system looks for the `filename` file in the simulation run directory. To access a file outside the simulation run directory, use a full file system pathname. If the `filename` file does not exist, *no* error is generated.
`n! [n!filename]`	Same as `[! filename]`, except that the contents of the new file are *not* parsed, and square-bracketed expressions are not interpreted.
`n? [n?filename]`	Same as `[? filename]`, except that the contents of the new file are *not* parsed, and square-bracketed expressions are not interpreted.

The function is defined in `bin/si`.

You *cannot* modify this function.

Arguments

`l_filelist`	The `fileList` argument is a list of lists for the input and output file names.
`t_runDirName`	The `runDirName` argument is the name of the simulation run directory. It must be a string name.
`t_libName`	The `libName` argument is the name of the library containing the top-level cellview of the design.
`t_cellName`	The `cellName` argument is the cell name of the top-level cellview.
`t_viewName`	The `viewName` argument is the view name of the top-level cell.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Examples

```
simInWithArgs( list("control" "si.inp") simRunDir
       simLibName simCellName simViewName )
```

```
simInWithArgs( '((infile1 outfile1)
   (infile2 outfile2))
   simRunDir simLibName simCellName
   simViewName )
```

simLoadNetlisterFiles

```
simLoadNetlisterFiles(
     t_fileName
     )
     => t / nil
```

Description

Loads the file given as an argument from the `local/fnl` directory. If it does not find the file in the `local/fnl` directory, it searches in the `etc/skill/fnl` directory in the install hierarchy. If it does not exist, no error is generated.

The function is defined in `etc/skill/si/simcap.ile`.

You can modify this function.

Arguments

t_fileName	Name of the file to be loaded.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simLoadNetlisterFiles( "silos.ile" )
```

simLoadSimulatorFiles

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

Description

Loads the simulator-specific file from the `local/si/caplib` directory. If it does not find the file in `local/si/caplib`, it searches `etc/skill/si/caplib` directory in the install hierarchy. If the file does not exist, *no* error is generated.

The function is defined in `etc/skill/si/simcap.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simLoadSimulatorFiles()
```

simNetExtNameCdsName

```
simNetExtNameCdsName(
     t_extNetName
     )
     => t_string
```

Description

Takes as an argument the netlister-assigned net name and returns the designer-assigned net name for `extNetName` as it appears in the schematic.

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

t_extNetName	Netlister-assigned net name

Value Returned

t_string	Returns the designer-assigned net name as it appears in the schematic. The exact name returned depends on the design.

Example

```
simNetExtNameCdsName( "N34" )
returns "/adder1/in"
```

simNetlistWithArgs

```
simNetlistWithArgs(
     t_simLibName
     t_simCellName
     t_simViewName
     t_simRunDir
     t_simNlpGlobalLibName
     t_simNlpGlobalCellName
     t_simNlpGlobalViewName
     l_simViewList
     l_simStopList
     t_simGlobalErrorFileName
     t_simProbeFileName
     t_simSimulator
     f_simTimeUnit
     f_simCapUnit
     t_simNetlistFileName
     t_simFlatViewTypeName
     )
     => t / nil
```

Description

Generates a flattened description of the design hierarchy specified by `simLibName`, `simCellName`, and `simViewName` in the syntax described by the global formatting properties in `simNlpGlobalLibName`, `simNlpGlobalCellName` and `simNlpGlobalViewNam`e.

In addition to specifying the input parameters, you must set the following variables from the SE global environment before calling this function:

```
simNetNamePrefix
simInstNamePrefix
simModelNamePrefix
```

These variables are used by `simNetlistWithArgs( )`.

The variables generate unique names during netlisting. The netlister uses these string prefixes and adds a unique number as a suffix to each of them to create a unique name. If you set these variables to `nil`, the netlister generates the unique number as output but does not add a string prefix to the number.

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

t_simLibName	The name of the library containing the top-level cellview of the design.
t_simCellName	The cell name of the top-level cellview (design) to be netlisted, for example, `"alu"` This cellname must be the `simCellName` global SE variable.
t_simViewName	The viewname of the top-level cellview of your design.
t_simRunDir	The full file system pathname to the simulation run directory. This pathname must be the same as the `simRunDir` global SE variable, for example, `"/mnt2/dave/simulations/silos1"`
t_simNlpGlobalLibName	The name of the library which contains the global cellview that defines the global format strings. The format strings define the netlist syntax. This library name must be the same as the `simNlpGlobalLibName` global SE variable, for example, `"basic"`
t_simNlpGlobalCellName	The cell name of the global cellview contains the global format strings that define the netlist syntax. This cell name must be the same as the `simNlpGlobalCellName` global SE variable, for example, `"nlpglobals"`
t_simNlpGlobalViewName	The view name of the global cellview that contains the global format strings that define the netlist syntax. This viewname must be the same as the `simNlpGlobalViewName` global SE variable, for example, `"verilog"`
l_simViewList	The view switch list that determines which view to switch into when a cell has been found. This list must be the same as the `simViewList` global SE variable, for example, `list("verilog" "schematic")`

l_simStopList	The stopping view list that determines when expansion of the hierarchy is halted. This list must be the same as the `simStopList` global SE variable, for example, `list("verilog")`
t_simGlobalErrorFile	The name of the file for storing global error messages.
t_simProbeFileName	The name of the probe file for storing errors, usually `probe.err`.
t_simSimulator	The simulator for which the netlist syntax is intended. This simulator must be the same as the `simSimulator` SE global environment variable, for example, `"verilog"`
f_simTimeUnit	The floating-point time unit factor, for example, `1.0e-9`.
f_simCapUnit	The floating-point capacitance unit factor, for example, `1.0e-15`.
t_simNetlistFileName	The name of the file in which the netlist is stored, for example, `"/mnt/dave/simulations/silos1/netlist"`

Value Returned

`t`	If the netlist process does not detect any errors, `t` is returned.
`nil`	If there is an error, `nil` is returned.

Example

```
simNetlistWithArgs(
   simLibName
   simCellName
   simViewName
   simRunDir
   simNlpGlobalLibName
   simNlpGlobalCellName
   simNlpGlobalViewName
   simViewList
   simStopList
   "global.err"
   "probe.err"
   simSimulator
   float(simTimeUnit)
   float(simCapUnit)
```

```
   "/mnt/dave/simulations/silos1/netlist
)
```

simNetCdsNameExtName

```
simNetCdsNameExtName(
     t_cdsNetName
     )
     => t_string
```

Description

Takes as an argument the designer-assigned net name for `cdsNetName` and returns the netlister-assigned net name as it appears in the netlist.

The function is defined in `bin/si`.

You *cannot* modify this function.

Arguments

t_cdsNetName	Designer-assigned net name in the schematic.

Value Returned

t_string	Returns the netlister-assigned net name as it appears in the netlist. The exact name returned depends on the design.

Example

```
simNetCdsNameExtName( "/adder1/in" )
returns "N34"
```

simNoNetlist

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

Description

Sets the `simNoNetlist` variable to `t` to specify that no netlist is generated, and then calls `sim` to perform a complete simulation, except for netlist generation.

The function is defined in `etc/skill/si/caplib/simulate.ile`.

You can modify this function.

Value Returned

`t`	Returns `t` on successful completion of `sim()`.
`nil`	Otherwise, returns `nil` and an error message is printed out.

Example

```
simNoNetlist()
```

simout

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

Description

Translates the netlister-assigned names for nets and instances in the `simout.tmp` file to the corresponding designer-assigned names and produces the `si.out` file. The `simout.tmp` file is normally the simulator text output. A message is printed stating that `simout` is being run, and then the `simOutWithArgs` function is called with the correct arguments to do the translation.

The following variables are used as arguments to the `simOutWithArgs` function. You must correctly set them before calling this function:

```
simSedFile
simRunDir
simLibName
simCellName
simViewName
```

All text in the `simout.tmp` file is copied to the `si.out` file, unless the text is surrounded by square brackets ([ ]). The opening square bracket ([ ) signals that the following text up to the closing square bracket ( ]) is to be interpreted. The entire expression is replaced by the resulting interpreted value. Following the opening square bracket ([ ) should be one of these command characters:

`#`	`[#netname]`	Replace the [#netname] expression with the designer-assigned net name for `netname`.
`$`	`[$instname}`	Replace the [$instname] expression with the designer-assigned instance name for `instname`.

Because names requiring translation are not output by the simulator surrounded by square brackets ([ ]), the interface developer must provide a `sed` input script for each simulator that surrounds each name requiring translation with square brackets ([ ]) and inserts the correct command character after the opening square bracket ([ ). For example, if the name `netname` needs to be translated back to the designer-assigned name for the net, the `sed` script needs to replace the word `netname` with `[#netname]` so that this function translates it.

The function is defined in `etc/skill/si/caplib/siminout.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` upon successful completion.
`nil`	Otherwise, returns `nil`.

Example

```
simout()
```

simOutWithArgs

```
simOutWithArgs(
     l_fileList
     t_runDirName
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Translates the netlister-assigned names for nets and instances in the input files specified in `fileList` to the corresponding user-assigned names, and produces the output files specified in `fileList`.

Because names requiring translation are not output by the simulator surrounded by square brackets ([ ]), the you must provide a `sed` input script for each simulator that surrounds each name to be translated with square brackets ([ ]). You must also insert the correct command character after the opening square bracket ([ ).

When the input arguments are processed the following conditions apply:

- If `libName` is `nil`, then the `simLibName`, `simCellName`, `simViewName` environment variables are used.
- Otherwise, if `cellName` is `nil`, then the `simCellName` environment variable is used.
- If `viewName` is `nil`, then the `simViewName` environment variable is used.

Therefore, the `libName` argument must not be `nil` if the `cellName` and `viewName` arguments are used (not `nil`).

All text in the `inputFileName` file is copied to the `outputFileName` file, unless it is surrounded by square brackets ([ ]). The opening square bracket ([) specifies that the following text up to the closing square bracket (]) must be interpreted. The entire expression is replaced by the interpreted value.

One of the command characters below must follow the opening square bracket ([ ):

`#`	`[#netname]`	Replace the [#netname] expression with the designer-assigned net name for `netname`.
`$`	`[$instname]`	Replace the [$instname] expression with the designer-assigned instance name for `instname`.

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

l_fileList	A list of lists for the input, output, and `sed` file names. The `sed` file name can be `nil` and a single list can be used for only one input, one output, and one `sed` file.
t_runDirName	The name of the simulation run directory. This must be a string name.
t_libName	The library name of the top-level design.
t_cellName	The name of the top-level cell.
t_viewName	The name of the top-level view.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Examples

```
simOutWithArgs(
   list ( infile outfile simSedFile)
   simRunDir
   simLibName
   simCellName
   simViewName
)


simOutWithArgs(
   '(( infile1 outfile1 simSedFile)
   ( infile2 outfile2 simSedFile))
   simRunDir
   simLibName
   simCellName
   simViewName
)
```

simPrintEnvironment

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

Description

Writes the primary simulation control variables and their values to the `si.env` file in the simulation run directory.

The following variables must be defined and are written to the file:

```
simLibName
simCellName
simViewName
simSimulator
simNotIncremental
simReNetlistAll
```

In addition, the following variables are written to the `si.env` file if they are defined:

```
simViewList
simStopList
simOtherInfo
simHost
```

The following variables are written to the `si.env` file if they are not `nil`:

```
simNetlistHier
simHostDiffers
simNoSimDiff
```

The variables specified by the `simSimulatorSaveVars` variable are also written to the file.

The function is defined in `etc/skill/si/caplib/init.ile`.

You can modify this function.

Arguments

None

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil` and prints an error message.

Example

```
simPrintEnvironment()
```

simPrintError

```
simPrintError(
     t_text
     )
     => t
```

Description

Prints the `text` argument to the *`stderr`* port if executed in the Cadence nongraphic environment. If the function is called from within the Cadence graphics environment, the output is written to the CIW window.

You can use this function instead of the following `fprintf` function:

```
fprintf( stderr text )
```

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

`t_text`	Text string

Value Returned

`t`	Returns `t` upon completion.

Example

```
simPrintError( "Can't open file simout.tmp\n" )
```

simPrintErrorLine

```
simPrintErrorLine(
     g_listArg
     )
     => t
```

Description

This function is used to list the terminals for a cell and write them out to a file.

Arguments

g_listArg	The following are the components of the list: ■ *t_libName*: Name of the library containing the design ■ *t_cellName*: Cell name of the design ■ *t_viewName*View name of the design ■ *t_fileName*: t for stdout (otherwise the file name)

Value Returned

t	Terms were printed successfully.

Example

```
simPrintErrorLine(l_listArg)
```

simPrintTermList

```
simPrintTermList(
     t_libName
     t_cellName
     t_viewName
     t_fileName
     )
     => t
```

Description

This function is used to list the terminals for a cell and write them out to a file.

Arguments

t_libName	Name of the library containing the design
t_cellName	Cell name of the design
t_viewName	View name of the design
t_fileName	't' for stdout (otherwise the file name).

Value Returned

t	Returns t upon completion.

Example

```
simPrintTermList("basic" "VDD" "schematic" "fileName")
```

simPrintMessage

```
simPrintMessage(
     t_text
     )
     => t
```

Description

Prints the `text` argument to the *stdout* port if executed in the Cadence nongraphic environment. If the function is called from within the Cadence graphics environment, the output is written to the CIW window.

You can use this function instead of the following `fprintf` function:

```
fprintf( stdout text )
```

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

t_text	Text string.

Value Returned

`t`	Returns `t` upon completion.

Example

```
simPrintMessage( "Can't open file simout.tmp\n" )
```

simReadNetCapFile

```
simReadNetCapFile(
     filename
     )
     => t / nil
```

Description

Reads in the net capacitance file and initializes the simAllNets global variable. This contains the list of all nets in the design and their associated capacitance values, which are present in the capacitance file.

Arguments

filename	It is the filename if the file is in the current directory or it can be full path to the file which contains the information about the nets and their associated capacitance. Provide the filename or full path to file which are valid SKILL strings. The default value is `<rundir>/lperun/sim.cap`

Value returned

`t`	Returns `t` upon successful reading of the file.
`nil`	Otherwise, returns nil.

Example

```
simReadNetCapFile( "./mydir/dir2/sim.cap" )
```

simRunDirInfile

```
simRunDirInfile(
     t_fileName
     )
     => t / nil
```

Description

Opens the `fileName` file in the simulation run directory for reading. It creates a full file system pathname to the file in the simulation run directory and then passes the file as an argument to the SKILL `infile` function. The function returns a SKILL `port`.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

t_fileName	Name of the file opened for reading.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simRunDirInfile( "tmp.in" )
```

simRunDirLoad

```
simRunDirLoad(
     t_fileName
     )
     => t / nil
```

Description

Loads the *t_fileName* file in the simulation run directory. It creates a full file system path to the file in the simulation run directory and then passes the file as an argument to the SKILL `load` function.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

t_fileName	Name of the file to be loaded.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simRunDirLoad( "comp.env" )
```

simRunDirOutfile

```
simRunDirOutfile(
     t_fileName
     )
     => t / nil
```

Description

Opens the `fileName` file in the simulation run directory for writing. It creates a full file system pathname to the file in the simulation run directory and then passes the file as an argument to the SKILL `outfile` function. The function returns a SKILL `port`.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

t_fileName	Name of the file to be opened.

Value Returned

`t`	Returns `t` upon successful completion of the command.
`nil`	Otherwise, returns `nil`.

Example

```
simRunDirOutfile( "tmp.out" )
```

simSetDef

```
simSetDef(
     variableName
     g_value
     )
     => t
```

Description

Sets the `variableName` variable to `value` only if the variable is not yet set or if the symbol `variableName` evaluates to `nil`. If the variable was set prior to calling this function, and the `simGenWarnings` variable is not `nil`, a warning message is printed to `stdout` that the value of the `variableName` variable is overridden.

The function is defined in `etc/skill/si/caplib/util.ile`.

You can modify this function.

Arguments

variableName	Name of the SE variable.
g_value	Value to be assigned to the SE variable.

Value Returned

`t`	Returns `t` upon successful completion.

Example

```
simSetDef('simCapUnit 1.0e-15)
```

simSetDefWithNoWarn

```
simSetDefWithNoWarn(
     variableName
     g_value
     )
     => t / nil
```

Description

Same as `simSetDef` except that no warning is generated if `variableName` is already set. The function is defined in `etc/skill/si/caplib/util.ile`. You can modify this function.

Arguments

variableName	Name of the SE variable.
g_value	Value to be assigned to the SE variable.

Value Returned

`t`	Returns `t` upon successful completion.
`nil`	Otherwise, returns nil.

Example

```
simSetDefWithNoWarn('simCapUnit 1.0e-15)
```

simStringsToList

```
simStringsToList(
     t_stringArg
     )
     => list / nil
```

Description

Function takes as an argument a string of names separated by blanks and tab characters and returns a list of strings. The function is defined in `bin/si`.

You *cannot* modify this function.

Arguments

t_stringArg	String of names separated by blanks and tab characters.

Value Returned

list	Returns a list of strings.
`nil`	Otherwise, returns `nil`.

Example

```
simStringsToList( "@ . /cds/etc/sdalib/schema" )
returns:
'("@" "." "/cds/etc/sdalib/schema")
```

simSubProbeCapByName

```
simSubProbeCapByName(
     netName
     )
```

Description

Display net capacitance by name.

Arguments

netName	String value to be given to functions which represents the name of the net for which the capacitance value is required.

Value Returned

None

Note: If the net name is valid, the capacitance value will be displayed on the net., else capacitance values for all the nets in the current design window will not be displayed.

Example

```
simSubProbeCapByName( "net16" )
```

simSubProbeCapByScreen

```
simSubProbeCapByScreen(
     )
```

Description

Display all the net capacitance on the current screen.

Arguments

None

Value returned

None

Example

```
simSubProbeCapByScreen()
```

simVertToHoriz

```
simVertToHoriz(
     t_inputFileName
     t_outputFileName
     )
     => t / nil
```

Description

Reads in the `inputFileName` file, converts names in vertical table headers to horizontal, and copies the rest of the file.

For example, if the input lines in the *t_inputFileName* file are

```
N N
4N4
234
6 7
```

they are printed in the *t_outputFileName* output file as

```
N 4 2 6
| N 3 |
N 4 4 7
```

The function is defined in `bin/si` and also in the Cadence graphics program. You *cannot* modify this function.

Note: This function is used only to generate SILOS and System HILO™ output files.

Arguments

t_inputFileName	Name of input file containing names for conversion.
t_outputFileName	Name of output file containing converted names.

Value Returned

`t`	Returns `t` on successful completion.
`nil`	Otherwise, returns `nil`.

Example

```
simVertToHoriz( "simout2.tmp" "simout.tmp" )
```

SE Graphics Functions

The following functions, which are defined by SE, simplify integrating your simulator into the Cadence graphics environment. These functions represent the functionality available on the Simulation menu. If you want to create your own menus and/or forms for the Simulation user interface, you can use them to perform the required SE functionality.

Note: These functions are defined only in the Cadence graphics environment and cannot be executed in the SI program. Because these functions cannot be modified, their descriptions do not include a “Defined in” section.

simCleanRun

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

Description

Deletes files created by both SE and the analysis tool being used from the simulation run directory. `simCleanRun` deletes only files that can be recreated by renetlisting or resimulating. Files which are required to rerun the simulation, such as the `si.env` and `control` files, are not deleted. The function displays a dialog box so you can confirm that you intend to delete the information. If the deletion is confirmed, the files that SE creates are deleted along with the files specified by the `simCleanFileList` variable. The `simCleanFileList` variable is set differently by each application integrated into SE.

Arguments

None

Value Returned

`t`	On successful completion.
`nil`	Otherwise, returns nil.

Example

```
simCleanRun( )
```

simEditFileWithName

```
simEditFileWithName(
     t_fileName
     )
     => ipcId / nil
```

Description

This function internally calls the `edit` function to display the specified file.

Arguments

t_fileName	Name of the file.

Value Returned

ipcId	This is the Id of the process initiated by call to MPS to view the file in VI.
`nil`	Otherwise, returns `nil`.

Example

```
simEditFileWithName("/tmp/skill.il")
```

simInitEnv

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

Description

Initializes the simulation environment within the Cadence graphics environment.

In addition to defining the SKILL environment needed for SE and the target application, the run directory is created and initialized as needed, using the `simInitRunDir` function. When `simInitEnv` is invoked, a form appears, prompting you for the simulation run directory name. You can specify either a relative or a full file system path name. If you specify a relative path name, the name is prepended with the full file system path name to the directory from which the program was invoked. If the specified run directory does not exist, a second form appears and prompts you for

- Simulation run directory
- Simulator name
- Library name
- Cell name
- View name

Using this information, the run directory is created and initialized. If the run directory already exists, the environment specified within it is restored, and the second form is not displayed.

Arguments

None

Value Returned

`t`	On successful completion.
`nil`	Otherwise, returns nil.

Example

```
simInitEnv( )
```

simInitEnvWithArgs

```
simInitEnvWithArgs(
     t_runDirName
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_simulatorName | nil
     g_forceInit
     )
     => t / nil
```

Description

Initializes the simulation environment within the Cadence graphics environment. In addition to defining the SKILL environment needed for SE and the target application, the run directory is created and initialized as needed.

Arguments

t_runDirName	The name of the simulation run directory to use.
t_libName	The name of the library containing the top-level cellview of the design to analyze.
t_cellName	The cell name of the top-level cellview of the design to analyze.
t_viewName	The view name of the top-level cellview of the design to analyze.
t_simulatorName	The name of the analysis tool to use.
g_forceInit	If the function returns `t`, and the specified run directory exists but is not initialized, the run directory is initialized with the specified parameters. If the function returns `nil`, there is an error.

The arguments can be used to overwrite the simulation environment variables. If you know that the run directory exists, all arguments except for *t_runDirName* can be `nil`. The contents of the run directory are then used to initialize the environment.

The `si.env` file is used to initialize a run directory for storing the values of the simulation environment variables. The arguments above can be used to change the simulation environment variables before the run directory is initialized. The arguments are processed according to the following rules:

- If the specified run directory does not already exist, the argument is not used, and the current graphics environment library path is stored in the new run directory.
- If *t_simulatorName* is not nil, assign it to the simSimulator environment variable.
- If *t_libName* is not nil, assign it to the simLibName environment variable.
- If *t_cellName* and t_viewName are not nil, assign them to the simCellName and simViewName environment variables, respectively.

Therefore, the *t_libName* argument cannot be nil if the *t_cellName* or *t_viewName* arguments are used (not nil).

If the si.env file exists in the run directory, it is loaded first, followed by the steps above. This will override what is stored in the run directory. Otherwise, the steps above are applied first, and the si.env file is created. Therefore, it is important that you determine the existence of the run directory before calling the simInitEnvWithArgs function. Parameters should be passed in if the directory does not exist, but should normally not be passed if it does exist.

Value Retuned

t	On successful completion.
nil	Otherwise, returns nil.

Example

```
simInitEnvWithArgs( "/mnt/dave/chip1/spice.run1"
"myLib" "fast_mux" "schematic"
"spice" nil )
simInitEnvWithArgs(    "/mnt/dave/chip1/spice.run1"
nil nil nil nil nil )
```

simPostNameConvert

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

Description

Frees all allocated storage and marks the map structure as invalid. Since this function is callable from SKILL, the parameter has to be maintained as a list.

Arguments

None

Value Returned

`t`	Returns t when the command is successful.
`nil`	Returns `nil` when the command is not successful.

Example

```
simPostNameConvert()
```

simPreNameConvert

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

Description

This function is called to set up variables needed for name conversion routines. If no arguments are passed, it does the setup by doing a lookup simulator environment settings.

Arguments

None

Value Returned

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

Example

```
simPreNameConvert()
```

simJobMonitor

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

Description

Displays a form listing the analysis jobs invoked in the background using the `simRunNetAndSim( )` and `simRunNetAndSimWithArgs( )` functions. The analysis job is listed on the form, along with its current status, time of invocation, and execution priority. Using this form, you can view the run log of a job, terminate the execution of an active job, suspend the execution of an active job, change the execution priority of a job, or delete a job from the form.

Arguments

None

Value Returned

`t`	On successful completion.
`nil`	Otherwise, returns nil.

Example

```
simJobMonitor( )
```

simRunNetAndSim

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

Description

Starts an analysis job in either foreground or background mode.

The simulation environment *must* be initialized before `simRunNetAndSim` is called. That is, the `simInitEnv` or `simInitEnvWithArgs` function must have been called. When invoked, `simRunNetAndSim` displays a form, prompting for the following:

- Simulation run directory (read only field)
- Library name
- Cell name
- View name
- Simulator name
- Whether to run the netlister
- Whether to run the simulator
- Whether to run in background/foreground
- Priority of a background simulation (read only if foreground)

`simRunNetAndSim` applies the form and invokes the simulation.

Arguments

None

Value Returned

`t`	Returns `t` if the form prompting for run options was successfully displayed.
`nil`	Otherwise, returns `nil`.

Example

```
simRunNetAndSim( )
```

simRunNetAndSimWithArgs

```
simRunNetAndSimWithArgs(
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_simulatorName | nil
     g_doNetlist
     g_doSimulation
     g_runBackground
     x_jobPriority
     )
     => t / nil
```

Description

Starts an analysis job in either foreground or background mode.

The simulation environment *must* be initialized before `simRunNetAndSim` is called. Specifically, you must call the `simInitEnv` or `simInitEnvWithArgs` function before calling `simRunNetAndSim`. The `simRunDir` global variable specifies the current run directory. It is set when the simulation environment is initialized.

Arguments

t_libName	The name of the library containing the top-level cellview of the design to analyze.
t_cellName	The cell name of the top-level cellview of the design to analyze.
t_viewName	The view name of the top-level cellview of the design to analyze.
t_simulatorName	The name of the analysis tool.
g_doNetlist	`t / nil`. If set to `t`, then the netlist for the design is generated.
g_doSimulation	`t / nil`. If set to `t`, the design is simulated. The simulator and the name translation functions are invoked. This invokes the same steps and functions as the `sim( )` function.
g_runBackground	`t / nil`. If set to `t`, the background process invokes the `bin/si` program to perform the simulation.
x_jobPriority	Priority of the background job (0 to 20). This is the UNIX priority that invokes the process; therefore, the lower the number, the higher the priority.

The *t_libName*, *t_cellName*, *t_viewName*, and *t_simulatorName* arguments overwrite the corresponding simulation environment variables. Specifying these arguments redefines the global environment *and* changes the values in the simulation run directory. If you want to use the current global environment, use `nil` as the value for these parameters. The arguments are processed according to the following rules:

- If *t_libName* is not `nil`, assign it to the `simLibName` environment variable.
- If *t_simulatorName* is not `nil`, assign it to the `simSimulator` environment variable.
- If *t_cellName* or *t_viewName* are not `nil`, assign them to the `simCellName` and/or `simViewName` environment variables, respectively.

Therefore, the *t_libName* argument must not be `nil` if the *t_cellName* and/or *t_viewName* arguments are used (not `nil`).

Value Returned

`t`	Returns `t` if the background process was successfully invoked. If the analysis is run in the foreground, `t` is returned if the analysis completed.
`nil`	Returns `nil` if the background process or foreground analysis failed.

Note: For this function, a return value of `t` does *not* necessarily mean that the analysis was completed successfully.

Example

```
simRunNetAndSimWithArgs( "myLib" "fast_mux"
"schematic" "spice" t t t 10)
   simRunNetAndSimWithArgs( nil nil nil nil
   t t t 10)
```

simRunNetAndSimWithCmd

```
simRunNetAndSimWithCmd(
     t_libName | nil
     t_cellName | nil
     t_viewName | nil
     t_simulatorName | nil
     t_cmdToBeExecuted
     g_runBackground
     x_jobPriority
     )
     => t / nil
```

Description

Executes a command you specify in either foreground or background mode. The simulation environment *must* be initialized before `simRunNetAndSimWithCmd` is called. The `simRunDir` global variable specifies the current run directory. It is set when the simulation environment is initialized.

Arguments

t_libName	The name of the library containing the top-level cellview of the design to analyze.
t_cellName	The cell name of the top-level cellview of the design to analyze.
t_viewName	The view name of the top-level cellview of the design to analyze.
t_simulatorName	The name of the analysis tool.
t_cmdToBeExecuted	The string specifying the name of the function to be executed.
g_runBackground	`t / nil`. If set to `t`, the background process performs the specified command.
x_jobPriority	Priority of the background job.

The *t_libName*, *t_cellName*, *t_viewName* and *t_simulatorName* arguments overwrite the corresponding simulation environment variables. Specifying these arguments redefines the global environment *and* changes the values in the simulation run directory. If you want to use the current global environment, use `nil` as the value for these parameters. The arguments are processed according to the following rules:

- If *t_libName* is not `nil`, assign it to the *simLibName* environment variable.

- If *t_simulatorName* is not nil, assign it to the simSimulator environment variable.
- If *t_cellName* and/or *t_viewName* are not nil, assign them to the simCellName and/or simViewName environment variables, respectively.
- Thus, the *t_libName* argument must not be nil if the *t_cellName* and/or *t_viewName* arguments are used (not nil).

Value Returned

t	Returns t if the background process was successfully invoked. If the analysis is run in the foreground, t is returned if the analysis completed.
nil	Returns nil if the background process or foreground analysis failed.

Note: For this function, a return value of t does *not* necessarily mean that the analysis was completed successfully.

Example

```
simRunNetAndSimWithCmd( "myLib"
   "fast_mux" "schematic"
   "spice" "simin" t 10
   )
simRunNetAndSimWithCmd( nil nil nil nil
   "my_SKILL_function" nil 10
   )
```

simViewFileWithArgs

```
simViewFileWithArgs(
     t_fileName
     l_windowSize
     [ t_windowTitle ]
     )
     => winID / nil
```

Description

This function calls the `view` function with the file name as the first argument and the window size as the second argument. The `view` function creates a viewFile window where the file text is displayed with the banner as the value supplied through the optional argument in the function `simViewFileWithArgs`.

Arguments

t_fileName	Name of the file.
l_windowSize	Size of the window. Specify the list containing the window size and maxWindowSize.
t_windowTitle	Title of the window.

Value Returned

winId	This is the Id of the window in which the file is displayed.
`nil`	Otherwise, returns `nil`.

Example

```
simViewFileWithArgs( "/tmp/skill.il" list(680:800 hiGetMaxScreenCoords()) "SKILL
CODE" )
```

simWaveOpen

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

Description

Invokes a form displaying the current simulation run directory name, and prompts you for the name of the waveform file to display. The default waveform file is `raw/waves` in the simulation run directory. When the file name is specified, a new window is opened, displaying the waveform information.

Arguments

None

Value Returned

`t`	On successful completion.
`nil`	Otherwise, returns nil.

Example

```
simWaveOpen( )
```

Note: WSF is no longer supported, therefore, this function will be removed in the future release of the product.

ISE Functions

This section describes the functions defined in the Interactive Simulation Environment (ISE).

iseCloseSchWindow

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

Description

Closes the schematic window.

Arguments

None

Value Returned

`t`	If the window is closed successfully.
`nil`	Returns `nil` with the error message in the log file.

Example

```
iseCloseSchWindow()
```

iseCloseSimWindow

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

Description

Closes the simulation window.

Arguments

None

Value Returned

`t`	If the window is closed successfully.
`nil`	Returns `nil` with the error message in the log file.

Example

```
iseCloseSimWindow()
```

iseCompleteInteractive

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

Description

Completes an interactive simulation session by closing all windows opened for interactive simulation and updates the ISE state machine so that ISE is aware that the interactive session has ended.

Arguments

None

Value Returned

`t`	If all the windows opened during interactive simulation are closed successfully.
`nil`	Returns `nil` with the error message in the log file.

Example

```
iseCompleteInteractive()
```

iseCommToSimulator

```
iseCommToSimulator(
     t_command
     )
     => t / nil
```

Description

Sends the command specified as the text parameter to the simulator. This function should be used in conjunction with input filtering if input filtering is in effect.

iseEnterNodeNamesList

```
iseEnterNodeNamesList(
     iseEnterPointFunc
     )
     => t / nil
```

Description

Prompts the designer to enter node names into a form and returns the netlister-assigned names as a SKILL list of strings corresponding to the schematic names entered.

If the designer enters nothing in the form, the function returns `nil`.

Arguments

iseEnterPointFunc	Valid SKILL function to be used as a callBack.This function is registered by the `iseEnterNodeNamesList()` function.

Value Returned

`t`	If the selection is a success.
`nil`	Returns `nil` with the error message in the log file.

Example

```
iseEnterNodeNamesList( 'myfunc )
```

iseExitSimulator

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

Description

Terminates the simulation using the command string you define through the ISE variable `iseExitSimulatorCommand`. If the simulation is remote, this routine copies every file from the remote host back to the local simulation run directory. To use the `iseExitSimulator` function properly, write a function in which you call `iseExitSimulator`. After this function returns, you can do post-processing such as translating netlister-assigned names.

If there is no simulator, or it is determined not to be running, this function returns `nil`. This function also returns `nil` if it is a remote simulation, but files are not successfully copied back.

Arguments

None

Value Returned

`t`	If simulator is successfully terminated.
`nil`	Returns `nil` with the error message in the log file.

Example

```
iseExitSimulator()
```

iseGetExtName

```
iseGetExtName(
     iseEnterPointsFunc
     )
     => t / nil
```

Description

Requires the call back function to be called by the enter points function in `iseGetExtName`. `The iseGetExtName` routine returns the netlister-assigned name of the object being pointed at in the schematic window.

If a net, terminal, or instance is not selected, or the name of the object cannot be translated, the function returns `nil`.

Arguments

iseEnterPointsFunc	Valid SKILL function to be used as a callBack.This function is registered by the `iseGetExtName()` function.

Value Returned

`t`	If the selection is a success.
`nil`	Returns `nil` with the error message in the log file.

Example

```
iseGetExtName( 'myfunc )
```

iseGetInputFromEncapWindow

```
iseGetInputFromEncapWindow (
     t_tmpStream
     )
     => t / nil
```

Description

This procedure is registered to be called whenever something is typed into the input window of the encapsulation window.This input - tmpStream - is passed into the procedure to be packaged before it is sent to the simulator. This routine also checks if the user has registered a function to be called so that this simulator inputs can be passed to this user-registered function for filtering. After filtering, the user can call the routine isePrintSimulatorCommand such that the data will be sent to the simulator. All inputs typed into the encapsulation window is automatically reflected in the history section of the encapsulation window. If no user-function is registered then the inputs are sent directly to the simulator without filtering.

Arguments

t_tmpStream	Valid SKILL string which consists of commands to be packaged and sent to the simulator.

Value Returned

`t`	Returns `t` if the commands were successfully sent to the simulator.
`nil`	Otherwise, returns `nil` with the error message in the log file.

Example

```
iseGetInputFromEncapWindow( "-v file.v -l run.log " )
```

iseGetMappedProbeList

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

Description

Returns a SKILL list of the netlister assigned names of all of the nets currently probed in the schematic window.

Arguments

None

Value Returned

`t`	If it can return the list of internally mapped names of current probe.
`nil`	Otherwise, returns `nil`.

Example

```
iseGetMappedProbeList()
```

iseGetProbeList

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

Description

Returns a SKILL list of the designer assigned names of all nets currently probed in the schematic window.

Arguments

None

Value Returned

`t`	If it can return the list of internally mapped names of current probe.
`nil`	Otherwise, returns `nil`.

Example

```
iseGetProbeList()
```

iseInitSchematicWindow

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

Description

Reads in the design specified by the `simLibName`, `simLibConfigName`, `simCellName`, `simViewName`, and `simVersionName` variables in the schematic window maintained by ISE.

Arguments

None

Value Returned

`t`	If it can initialize the simulator window for the current session.
`nil`	Otherwise, returns `nil` with the error message in the log file.

Example

```
iseInitSchematicWindow()
```

iseInitSimWindow

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

Description

Initializes the simulator window.

First, the current window is set as the simulation window. Next, the menu for that window is set as the menu whose menu handle is specified by the `iseSimulatorMenuHandle` variable. No simulator menu will appear in the window unless this variable is set. Then, a check is performed to see if there is a netlist in the simulation run directory. If there is none, you are prompted to choose whether you want to netlist. A new netlist is generated and the simulator is invoked if you specify *yes*. If you specify *no*, no netlist is generated, and the simulator is not invoked but the process remains inside the interactive simulation environment. This function returns `nil` if the simulator window is not accessible, if a simulator is running, or if the simulator is not invoked successfully; otherwise, this function returns `t`.

Arguments

None

Value Returned

`t`	If it can initialize the simulator window for the current session.
`nil`	Otherwise, returns `nil` with the error message in the log file.

Example

```
iseInitSimWindow()
```

iseInterruptSimulator

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

Description

Issues a soft interrupt to the simulator. If a simulator window is accessible, the function returns `t`; otherwise, it returns `nil`.

iseNetExtNameCdsName

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

Description

Brings up a form for the designer to enter a netlister-assigned node name. A second form appears which displays both this name and its corresponding designer-assigned name.

Arguments

None

Value Returned

`t`	If the form is successfully opened for the user.
`nil`	Otherwise, returns `nil`.

Example

```
iseNetExtNameCdsName()
```

iseOpenWindows

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

Description

Opens three new windows and updates internal ISE structures required to keep track of the identifications of these windows. The first covers the bottom right quarter of the screen and is set to be the schematic window. The second occupies the upper half of the screen and is the waveform window. The third window occupies the lower left quarter of the screen and is used to run the simulator. The original windows are not altered, but instead are overlaid with the new windows. This is the default window configuration.

If the `iseDontOpenSchematicWindowIfOneExists` variable is set to `t`, ISE searches all windows and makes the ISE schematic window the first window it finds with the appropriate design that was opened in append mode. If no such window exists, ISE searches all windows and makes the ISE schematic window the first window it finds with the appropriate design that was opened in read or write mode. If no such window is found, ISE opens a new schematic window. If the `iseNoWaveformWindow` is set to `t`, ISE does not open a waveform window. In this case, the designer must open a waveform window if it is needed.

Arguments

None

Value Returned

`t`	If all the windows are successfully opened.
`nil`	Otherwise, returns `nil`.

Example

```
iseOpenWindows()
```

isePrintName

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

Description

Prints the netlister assigned name of the object being pointed at in the schematic window to the input portion of the simulation window, starting from the current cursor position.

Arguments

None

Value Returned

`t`	Returns `t` if the window is accessible.
`nil`	Otherwise, returns `nil`.

Example

```
isePrintName()
```

isePrintNameCB

```
isePrintNameCB(
     t_tempList
     )
     => t / nil
```

Description

This function is the callback function of `isePrintName` which is to be called by the `enterPoints` function in `iseGetExtName`.

After receiving the mapped name(s), this function print the name(s) of the object selected from the schematic to the command line in the input section of the encapsulation window.

Arguments

t_tempList	The list of strings which are mapped names of the objects selected from the schematic window.

Value Returned

`t`	Returns `t` if command is successful.
`nil`	Returns `nil` if an error occur and the error is placed in the log file.

Example

```
isePrintNameCB( '("a" "b" "c" ) )
```

isePrintSimulatorCommand

```
isePrintSimulatorCommand(
     [ commandArg | nil ]
     )
     => t / nil
```

Description

This function is used when you create menu commands that interface with both the design and simulator.

For example, it makes it possible to instruct the simulator to set a node by pointing at the node in the design. The SKILL function underlying the menu entry determines the name of the node pointed in the design, translates the name to the name assigned by the netlister, and issues the appropriate command to set the node with the determined name to the simulator.

Arguments

commandArg	The argument `commandArg` is optional. If no argument is passed, then the function will open a form for the user to enter the command to be passed to the simulator.

Value Returned

`t`	Returns `t` if the window is accessible.
`nil`	Otherwise, returns `nil`.

Example

```
isePrintSimulatorCommand( "-f test -v verilog.v" )
```

iseReleaseNodeFrom

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

Description

After forcing a node to a preferred value, this function calls the function specified by the `iseReleaseFunc` variable to release the node from that value.

Arguments

None

Value Returned

`t`	Returns `t` if the command is successfully sent to the simulator.
`nil`	Otherwise, returns `nil`.

Example

```
iseReleaseNodeFrom()
```

iseSendOutputToEncapHistory

```
iseSendOutputToEncapHistory(
     t_myText
     )
     => t / nil
```

Description

This function writes the string passed, in as a parameter to the history section of the encapsulation window.

Arguments

t_myText	The argument can be a valid SKILL string, integer or a float value. The function will check for the type of an argument passed and append it to the history section of the encapsulation window.

Value Returned

`t`	Returns `t` when the function is successful in appending the text in the history.
`nil`	Otherwise, returns `nil` with the error in the log file.

Examples

- In case of a string value

```
iseSendOutputToEncapHistory( "Hello World")
```

- For integer value

```
iseSendOutputToEncapHistory( 10 )
```

- For float values

```
iseSendOutputToEncapHistory( 4.5 )
```

iseSearchForASchWindow

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

Description

If `iseDontOpenSchematicWindowIfOneExists` is set to `t`, this function searches for the first existing schematic window with the correct design opened for edit and use that window as the ISE schematic window.

If not available, this function searches for a window with the same design but opened for read only. If such a window is found, then the window is used as the ISE schematic window. If no such window is found, then it returns nil.

Arguments

None

Value Returned

`t`	Returns `t` if command is successful.
`nil`	Returns `nil` if error occur and the error message is placed in the log file.

Example

```
iseSearchForASchWindow()
```

iseSetEncapBindKeys

```
iseSetEncapBindKeys(
     )
```

Description

This function set the bindkeys for the encapsulation window.

Ensure that the bindkeys must be set before opening the window. The reason being setting bindkeys after the window is opened is an expensive operation in terms of memory and time.

Arguments

None

Value Returned

None

Example

```
iseSetEncapBindKeys()
```

iseSetNodeTo

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

Description

This function replaces the routine `iseSet`. It calls the function specified by the `iseSetFunc` variable.

Arguments

None

Value Returned

`t`	Returns `t` if the function specified by `iseSetFunc` variable is set and a simulation window is accessible.
`nil`	Otherwise, returns `nil`.

Example

```
iseSetNodeTo()
```

iseSimulate

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

Description

This function calls the function specified by the `iseSimulateFunc` variable.

Arguments

None

Value Returned

`t`	Returns `t`, if successful in calling the function set by the variable `iseSimulateFunc` and simulation window exists.
`nil`	Otherwise, returns `nil`.

Example

```
iseSimulate()
```

iseStartInteractive

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

Description

This function sets up the default interactive simulation environment.

Before you start an interactive simulation session, you must have initialized the simulation environment by executing the `Initialize` command, which can be found in the `Simulation` menu.

Three new windows are created which overlay any windows currently displayed. The creation and size of each window is controlled by global SKILL variables. The opening of windows is performed by executing the function specified by the `iseOpenWindowsFunc` variable.

One window displays waveforms. If a waveform file already exists in the run directory, the waveforms are read in, and the menu for that window is set to be the Waveform menu. Initialization of the waveform window is performed by executing the function specified by the `iseInitWaveWindowFunc` variable.

The second window displays the design. The top level of the design, as specified by the variables `simLibName`, `simLibConfigName`, `simCellName`, `simViewName`, and `simVersionName`, is automatically displayed.

The third window lets you interact with the simulator. To initialize this window, the function specified by the `iseInitSimWindowFunc` variable is executed. The default for this variable is the `iseInitSimWindow()` function. The simulator is automatically started and associated with this window. First, the function specified by `iseStartSimulatorFunc` is executed. The default value for this variable is the `iseStartSimulator()` function. This function ensures that the simulation window is available and evaluates the `iseInvokeSimulatorFunc` variable to invoke the simulator. If it is `nil` (that is, you do not define it), then the variable `iseRunSimulatorCommand` must be defined (the command string to invoke the simulator) so ISE can invoke your simulator. If the variable `iseInvokeSimulatorFunc` is set to the name of a function, that function is called so you can do any required preprocessing. Before the preprocessing function is called, the variable `iseRunSimulatorCommand` may or may not be defined. If it is not defined, it is expected to be defined before the preprocessing routine returns. The command string in `iseRunSimulatorCommand` is used by ISE to invoke your simulator.

The `iseInvokeSimulatorFunc` function takes precedence over `iseRunSimulatorCommand`. If both are defined at the beginning of the routine

`iseStartSimulator`, the variable `iseInvokeSimulatorFunc` is always evaluated first; that is, the preprocessing routine will be called. If the variable `iseInvokeSimulatorFunc` is undefined (with value `nil`) and the variable `iseRunSimulatorCommand` is defined, the command string in `iseRunSimulatorCommand` is used directly to invoke the simulator. If both are undefined, or the preprocessing function evaluates to `nil`, this function will return `nil` and the simulator will not be invoked.

The menu for this window is set to the menu whose menu handle is specified by the `iseSimulatorMenuHandle` variable.

Because this default initialization sequence may not suit every designer's needs, many features can be separately parameterized with global SKILL variables. Refer to the "ISE Variables" section for details on global variables.

Arguments

None

Value Returned

`t`	Returns `t`, if successful in starting interactive mode simulation.
`nil`	Otherwise, returns `nil`.

Example

```
iseStartInteractive()
```

iseStartSimulator

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

Description

This function issues the command to start the simulator in the simulator window by first evaluating the `iseInvokeSimulatorFunc` variable.

If it is `nil` (that is, you do not define it), the variable `iseRunSimulatorCommand` must be defined (the command string to invoke the simulator) so that ISE can invoke your simulator. If the variable `iseInvokeSimulatorFunc` is set to the name of a designer-defined function, that function is called so that you can do any preprocessing needed. Before the pre-processing function is called, the variable `iseRunSimulatorCommand` may or may not be defined. If it is not, it is expected to be defined before your pre-processing routine returns. The command string in `iseRunSimulatorCommand` is used by ISE to invoke your simulator. The `iseInvokeSimulatorFunc` function takes precedence over `iseRunSimulatorCommand`. If both are defined going into the routine `iseStartSimulator`, the variable `iseInvokeSimulatorFunc` is always evaluated first, that is, your pre-processing routine will be called. If the variable `iseInvokeSimulatorFunc` is undefined (with value `nil`) and the variable `iseRunSimulatorCommand` is defined, the command string in `iseRunSimulatorCommand` is used directly to invoke the simulator. If both are undefined or your pre-processing function evaluates to `nil`, this function will return `nil` and your simulator will not be invoked.

Arguments

None

Value Returned

`t`	Returns `t`, if able to start the simulator.
`nil`	Otherwise, returns `nil`.

Example

```
iseStartSimulator()
```

iseUpdateNetlist

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

Description

This function generates a new netlist by calling the SE `netlist` function.

If there were no errors, the command specified by the `iseInputNetlistCommand` variable is issued to the simulator if a simulator window exists and the command is not `nil`.

Arguments

None

Value Returned

`t`	Returns `t`, if able to netlist.
`nil`	Otherwise, returns `nil`.

Example

```
iseUpdateNetlist()
```

iseUpdateStimulus

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

Description

This function runs the input name translation `simin` function.

Then issues the command specified by the `iseInputStimulusCommand` variable to the simulator if a simulator window exists and the command is not `nil`.

Arguments

None

Value Returned

`t`	Returns `t`, in case able to run `simin()` successfully.
`nil`	Otherwise, returns `nil`.

Example

```
iseUpdateStimulus()
```

HNL Access Functions

This section describes property, database, and print SKILL functions of the Hierarchical Netlister (HNL).

Property Functions

Property functions can be used to search for properties and to scale property values.

hnlGetCellHdbProps

```
hnlGetCellHdbProps(
     d_cellView
     )
     => list / nil
```

Description

This function is called by the formatter during netlisting to get the list of HDB properties and their values for the given cellview. The function uses the DBId of cellview to extract the libName and cellName. It further uses the libName/cellName combination to query the HDB properties from `prop.cfg` property file.

Arguments

d_cellView	DBId of the cellView that is searched to get libName and cellName.

Value Returned

list	Returns the list of properties for the given cellview from the HDB.
nil	Returns when the command is not successful.

Example

```
hnlGetCellHdbProps( )
```

hnlGetSimulator

```
hnlGetSimulator(
     )
     => ams
```

Description

This function provides a distinction between AMS and other flows. When it is called from the AMS flow, the function displays AMS as output, otherwise it behaves the same as simSimulator.

Arguments

None

Value Returned

`ams`	Returns AMS when it is called from the AMS flow.

Example

```
if(hnlGetSimulator()=="ams" then hnlVerilogIgnoreTerm=nil)
```

hnlPcellIsParamOverridden

```
hnlPcellIsParamOverridden(
     dbobject
     t_string
     )
     => t / nil
```

Description

This function finds if the Pcell param of a master is overridden at the instance.

Arguments

dbobject	Specifies the database ID of the cellview that is searched to get the library name and the cell name.
t_string	Specifies the string to be searched

Value Returned

`t`	Returns `t` is the param is overriden
`nil`	Returns `nil` if the param is not overriden and an error if no related param is found

hnlGetPropVal

```
hnlGetPropVal(
     t_propName
     d_cellView
     d_inst
     )
     => propValue / nil
```

Description

Returns the value of the property whose string name is given as the first argument. The property is searched for first on the instance given as the third argument, and if not found there, on the cellview given as the second argument. If the property is found, the value is returned. Otherwise, nil is returned.

Arguments

t_propName	Name of the property to be retrieved.
d_cellView	propNameId of the cellView that is searched for the property value.
d_inst	instId of the instance that is searched for the property value.

Value Returned

propValue	Returns the value of the property upon successful completion.
nil	The command is not successful.

Example

```
hnlGetPropVal( "l" cellView inst )
```

hnlGetRoundProp

```
hnlGetRoundProp(
     t_propName
     )
     => propValue / nil
```

Description

Locates the property of the given name, using hnlGetPropVal( ), hnlCurrentMaster, and hnlCurrentInst, and then returns the result rounded to the nearest integer. If the named property is not found, nil is returned.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Arguments

t_propName	Name of the property retrieved.

Value Returned

propValue	Returns the value of the property rounded to the nearest integer
nil	The command is not successful.

Example

```
hnlGetRoundProp( "l" )
```

hnlGetScaleCapacitance

```
hnlGetScaleCapacitance(
     t_propName
     )
     => propValue / nil
```

Description

Locates the property of the given name, using `hnlGetPropVal`, `hnlCurrentMaster`, and `hnlCurrentInst`, divides the value by the value of the *`simCapUnit`* variable, and then returns the result rounded to the nearest integer. If the named property is not found, `nil` is returned. If `simCapUnit == nil`, the value of the property is returned, again rounded to the nearest integer.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Arguments

`t_propName`	Name of the property retrieved

Value Returned

`propValue`	Returns the value of the property divided by the *`simCapUnit`* variable. The returned value is rounded to the nearest integer.
`nil`	The command is not successful.

Example

```
hnlGetScaleCapacitance( "capValue" )
```

hnlGetScaleMarginalDelay

```
hnlGetScaleMarginalDelay(
     t_propname
     )
     => propValue / nil
```

Description

Locates the property of the given name, using `hnlGetPropVal( )`, `hnlCurrentMaster`, and `hnlCurrentInst`, divides the value by the value of the *`simTimeUnit`* variable, multiplies it by the value of the simCapUnit variable, and then returns the result rounded to the nearest tenth. If the named property is not found, nil is returned. If simTimeUnit == nil, the value of the property is returned, again rounded to the nearest tenth.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Argument

`t_propName`	Name of the property retrieved.

Value Returned

`propValue`	Returns the value of the property divided by the `simTimeUnit` variable and then multiplied by the `simCapUnit` variable. The returned value is rounded to the nearest tenth.
`nil`	The command is not successful

Example

```
hnlGetScaleMarginalDelay( "l" )
```

hnlGetScaleTimeUnit

```
hnlGetScaleTimeUnit(
     t_propName
     )
     => propValue / nil
```

Description

Locates the property of the given name, using `hnlGetPropVal`, `hnlCurrentMaster`, and `hnlCurrentInst`, divides the value by the value of the simTimeUnit variable, and then returns the result rounded to the nearest integer. If the named property is not found, nil is returned. If `simTimeUnit == nil`, then the value of the property is returned, again rounded to the nearest integer.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Argument

t_propName	Name of the property retrieved.

Value Returned

propValue	Returns the value of the property divided by the `simTimeUnit` variable. The returned value is rounded to the nearest integer.
`nil`	The command is not successful.

Example

```
hnlGetScaleTimeUnit( "l" )
```

hnlGetSourceFile

```
hnlGetSourceFile(
     d_instId | d_cellviewId
     )
     => path
```

Description

This function is called by the formatter during netlisting to get the path of source file of the given instance or cellview as specified in the HDB (*prop.cfg* property file).

Arguments

d_instId	instId of the instance.
d_cellViewId	cellviewId of the cellview.

Value Returned

path	Returns the path of the source file of the given cellview or instance.

Example

```
hnlGetSourceFile(hnlCurrentInst)
"/tmp/s1.v"

hnlGetSourceFile(hnlCurrentMaster)
"/tmp/cell.v"
```

hnlGetSourceFileModels

```
hnlGetSourceFileModels(
     )
     => listCells
```

Description

This function is called by the formatter during netlisting to get a list of names of the netlisted cells. For the cells that have the `-subckt` value set using the `sourcefile_opts` property, the cell names in the list are replaced by the subcircuit names, `-subckt <name>`.

Arguments

None

Value Returned

listCells	Returns a list of names of the netlisted cells.

Example

```
hnlGetSourceFileModels()
("cell1" "cell2" "subckt_opts_name" "cell4")
```

hnlGetSymbolPropVal

```
hnlGetSymbolPropVal(
     s_propSymbol
     d_cellView | d_inst
     )
     => propValue / nil
```

Description

Returns the value of the property whose symbol name is given as the first argument. The property is searched for first on the instance given as the third argument, and if not found there, on the cellview given as the second argument. If the property is found, the value is returned; otherwise, `nil` is returned.

Arguments

s_propSymbol	Symbol name of the property retrieved.
d_cellView	cellViewId of the cellView that is searched for the property value.
d_inst	instId of the instance that is searched for the property value.

Value Returned

propValue	Returns the value of the property whose symbol name is given as the first argument.
`nil`	Otherwise returns `nil`.

Example

```
hnlGetSymbolPropVal( "l" cellView inst )
```

hnlGetInstanceCount

```
hnlGetInstanceCount(
     )
     => numInstances
```

Description

Returns the number of instances netlisted in a single session. The function does not count the instances for which the `nlAction=ignore` property is set. This functions helps estimate the time taken to netlist and simulate based on the number of instances in a design. For example, this function can be used to implement a progress bar. A formatter can use this function only when OSS calls formatter functions to print a netlist.

Arguments

None

Value Returned

numInstances	Returns an integer value, which represents the number of instances netlisted in a single session.

Example

```
hnlGetInstanceCount()
```

hnlEMHGetDigitaGlobalNets

```
hnlEMHGetDigitaGlobalNets(
     )
     => globalNetList
```

Description

Returns a list of global nets found in digital design during mixed signal netlisting. This is used by auCdl netlister to print *.GLOBAL statements for these nets in the top level netlist. This function must be called from Cadence internal formatters or ADE third-party integration formatters only when mixed signal netlisting is in progress.

Arguments

None

Value Returned

globalNetList	List of global net names.

Example

```
hnlEMHGetDigitalGlobalNets()
=> ("vdd" "gnd")
```

hnlEMHGetDigitalNetlistFileName

```
hnlEMHGetDigitalNetlistFileName(
     )
     => path
```

Description

Returns the netlist file path for digital netlist during mixed signal netlisting. This path is exported for use by formatters like auCdl to include digital netlist file name in top level netlist. This function must be called from Cadence internal formatters or ADE third-party integration formatters only when mixed signal netlisting is in progress.

Arguments

None

Value Returned

path	The netlist file path.

Example

```
hnlEMHGetDigitalNetlistFileName()
> "runDir/top.cdl"
```

hnlEMHSetVerbosityLevel

```
hnlEMHSetVerbosityLevel(
     x_integer
     )
     => t
```

Description

Raises the level of informative output from mixed signal netlister. To avoid cluttering of output, the level should be set to one.

Arguments

x_iteger	Level of informative output.

Value Returned

t	The level of informative output has been set to the specified value.

Example

```
hnlEMHSetVerbosityLevel(1)
```

hnlOpenTopCell

```
hnlOpenTopCell(
     t_lib
     t_cell
     t_view
     )
     => obj
```

Description

This function opens the top level cell view. If lib/cell/view is an HDB config, then first open the config before opening the top level cell view.

Arguments

t_lib	Library name
t_cell	Cell name
t_view	View name

Value Returned

obj	SKILL object of the dbId for the cell view passed for opening is returned.

Example

```
topblock = hnlOpenTopCell( libName cellName viewName )
topblock will give the SKILL object for the cell view opened
```

hnlScaleCapacitance

```
hnlScaleCapacitance(
     g_propVal
     t_propName
     )
     => value
```

Description

Takes the value given as argument, divides it by the value of the simCapUnit variable, and then returns the result rounded to the nearest integer. If simCapUnit == nil, then the value of the property is returned, again rounded to the nearest integer. The property name given as the second argument is used for error messages if the value does not evaluate to a number.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Arguments

g_propVal	Value scaled.
t_propName	Name of the property.

Value Returned

value	Returns the value of the first argument divided by the `simCapUnit` variable. The returned value is rounded to the nearest integer. If `simCapUnit == nil`, then the value of the first argument is returned, again rounded to the nearest integer.

Example

```
hnlScaleCapacitance( 5 "1" )
hnlScaleCapacitance( "5" "1" )
```

hnlScaleMarginalDelay

```
hnlScaleMarginalDelay(
     g_propVal
     t_propName
     )
     => value
```

Description

Takes the value given as argument, divides the value by the value of the simTimeUnit variable, multiplies it by the value of the simCapUnit variable, and then returns the result rounded to the nearest tenth. If the named property is not found, nil is returned. If simTimeUnit == nil or simCapUnit == nil, the value of the property is returned, again rounded to the nearest tenth. The property name given as the second argument is used for error messages if the value does not evaluate to a number.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Arguments

g_propVal	Value scaled.
t_propName	Name of the property.

Value Returned

value	Returns the result of the first argument divided by the `simTimeUnit` variable and then multiplied by the `simCapUnit` variable. The returned value is rounded to the nearest tenth. If either `simTimeUnit` or `simCapUnit` is `nil`, the value of the first argument is returned, again rounded to the nearest tenth.

Example

```
hnlScaleMarginalDelay( 5 "l" )
hnlScaleMarginalDelay( "5" "l" )
```

hnlScaleTimeUnit

```
hnlScaleTimeUnit(
     g_propVal
     t_propName
     )
     => value
```

Description

Takes the value given as argument, divides it by the value of the simTimeUnit variable, and then returns the result rounded to the nearest integer. If simTimeUnit == nil, the value of the property is returned, again rounded to the nearest integer. The property name given as the second argument is used for error messages if the value does not evaluate to a number.

Note: If the property is of the type “string,” the property value is evaluated before it is scaled.

Arguments

g_propVal	Value of the property.
t_propName	Name of the property.

Value Returned

value	Returns the result of the first argument divided by the `simTimeUnit` variable. The returned value is rounded to the nearest integer. If `simTimeUnit` is `nil`, the value of the first argument is returned, again rounded to the nearest integer.

Example

```
hnlScaleTimeUnit( 5 "l" )
hnlScaleTimeUnit( "5" "l" )
```

Database Functions

Database functions provide information about the design and the internal data structures of HNL specific to netlisting.

hnlIgnoreTerm

```
hnlIgnoreTerm(
     d_term
     )
     => t / nil
```

Description

Returns t if the terminal argument belongs to a patchcord or to an instance whose master is not a stopping cell and does not contain any instances. This function excludes terminals attached to instances such as vdd and gnd.

Arguments

d_term	termId of the terminal.

Value Returned

t	Returns t if the terminal belongs to a patchcord or to an instance whose master is not a stopping cell and which contains no instances.
nil	The command is not successful.

Example

```
hnlIgnoreTerm( termA )
```

hnlIsAPatchCord

```
hnlIsAPatchCord(
     d_inst
     )
     => t / nil
```

Description

Returns t if the master of the given instance is a patchcord.

Arguments

d_inst	instId of the instance whose master is checked.

Value Returned

`t`	Returns `t` if the master of the given instance is a patchcord
`nil`	Returns `nil` when the command is not successful.

Example

```
hnlIsAPatchCord( inst )
```

hnlIsAStoppingCell

```
hnlIsAStoppingCell(
     d_cellView
     )
     => t / nil
```

Description

Returns t if the cellview argument is a stopping cellview. A cell is a stopping point for expansion if its viewName is in the hnlViewList switch view list as well as the hnlStopList stopping list, or if the cell has a string property nlAction with stop as its value. This function is used throughout the netlister to determine when to stop expansion and whether to output a macro reference or a reference to a primitive device.

Arguments

d_cellView	cellViewId of the cell view checked.

Value Returned

`t`	Returns `t` if the cell view given as the argument is a stopping cell.
`nil`	Returns nil when the command is not successful.

Example

```
hnlIsAStoppingCell( cellView )
```

hnlIsCurrentInstStopping

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

Description

Returns t if the current instance (hnlCurrentInst) is a stopping instance. An instance is a stopping point for expansion if the instance's Master is in the instance specific stopList. An instance is also a stopping point if the instance has a string property, nlAction with stop as its value. This function is used throughout the netlister to determine when to stop expansion and whether to output a macro reference or a reference to a primitive device.

Arguments

None

Value Returned

`t`	Returns t if the current instance is a stopping instance.
`nil`	Returns `nil` when the command is not successful.

Example

```
hnlIsCurrentInstStopping()
```

hnlMultipleCells

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

Description

Checks if the cellname argument exists for more than one cellname or cellview in the list of all cells. This function is used to detect cases of cells having the same name, but which are found in more than one location of the design hierarchy. This happens when cells with the same name are found in more than one library.

Arguments

t_name	Name of cell.

Value Returned

`t`	Returns `t` if the cellname argument is listed for more than one cellname or cellview in the list of all cells.
`nil`	Returns `nil` when the command is not successful.

Example

```
hnlMultipleCells( "inv" )
```

hnlNameOfSignal

```
hnlNameOfSignal(
     d_net
     x_netIndex
     )
     => name / nil
```

Description

Returns a string for the name of the signal that defines the given (net index) pair. This name is considered to be the preferred name and is consistent even when called with two different nets whose ~> name field differs, but which are electrically equivalent at this level of the design hierarchy. If the netIndex is not in the range of bits for the given net, nil is returned.

If this bit of net is an inherited net or is connected to an inherited terminal then the *netlister-generated* name is returned.

Arguments

d_net	netId of the net.
x_netIndex	Index of a bit of the net.

Value Returned

name	Returns the preferred name of the signal that defines the given (net index) pair.
nil	Returns if the netIndex is not in the range of bits for the given net.

Example

```
hnlNameOfSignal( net 0 )
```

The call to this function for signal *vdd!* in the *schematic* view of the cell *inv* (hnlNameOfSignal(db_id_signal_vdd!, 0)) returns inh_vdd - the local name. It does not return the inherited signal name since that depends on the hierarchy lineage

hnlNetNameOnTerm

```
hnlNetNameOnTerm(
     t_termName
     x_bit
     )
     => name / nil
```

Description

Returns the signal name of the net attached to the given bit of the terminal of the given name on the current instance being expanded. This function guarantees the same name is returned even if the net is aliased, and this function is later called with a terminal attached to one of the aliases of this net.

If this bit of net is connected to an inherited terminal then the *netlister-generated* name is returned.

Arguments

t_termName	Name of the terminal.
x_bit	Bit of the terminal attached to the returned signal name of the net.

Value Returned

name	Returns the signal name of the net attached to the specified bit of the specified terminal for the current instance being expanded.
nil	Returns if the terminal is not found or if the specified bit is not in the range of bits of the given terminal.

Example

```
hnlNetNameOnTerm( "IN<3:0>" 2 )hnlNetNameOnTerm( "IN" 0 )
```

hnlNetNameOnTermName

```
hnlNetNameOnTermName(
     t_termName
     )
     => name / nil
```

Description

Returns the signal name of the net attached to the terminal of the given name on the current instance being expanded. This function guarantees the same name is returned even if the net is aliased, and this function is later called with a terminal attached to one of the aliases of this net.

Arguments

t_termName	Name of the terminal

Value Returned

name	Returns the signal name of the net attached to the specified terminal for the current instance being expanded.
nil	Returns if the terminal is not found.

Example

```
hnlNetNameOnTermName( "IN<3:0>")
```

hnlAddExtraParameters

```
hnlAddExtraParameters(
     l_list
     )
     => t / nil
```

Description

Adds user-specified design variables to the OSS design variables list. For example, ADE-XL requires to add more design variables to OSS design variables list that are obtained after CDF callback evaluation.

These variables appear in control files created by OSS for internal use and therefore make sure that the design is not entirely re-netlisted on subsequent netlisting.

Arguments

l_list	List of strings, where each string is a parameter name

Value Returned

`t`	Indicates success
`nil`	Indicates failure; this function returns nil in the following cases: ■ this function is called when OSS is not generating a netlist ■ the input list is incorrectly formatted

Example

```
hnlAddExtraParameters((list "CAP0" "CAP1"))
```

hnlCellExtracted

```
hnlCellExtracted(
     d_cellView
     )
     => t / nil
```

Description

Checks that the cellview has not been modified since it was last extracted.

Arguments

d_cellView	cellViewId of the cell view checked.

Value Returned

t	Returns t if the given cell view has not been modified since it was last extracted.
nil	Returns nil when the command is not successful.

Example

```
hnlCellExtracted( cellView )
```

hnlCellInAllCells

```
hnlCellInAllCells(
     d_cellView
     )
     => t / nil
```

Description

Returns t if the cellview argument exists as a cellview in the list of all cells. If not, this function returns nil.

Arguments

d_cellView	The cellViewId of the cell view checked. If Hierarchical Configuration is used, the following arguments also should be supplied: ■ *t_viewList*: Effective view list ■ *t_pathName*: Path name string ■ *t_LibName*: Configuration library name ■ *t_cellName*: Cell name ■ *t_viewName*: View name ■ *t_isCellTopCell*: If this cell is top cell

Value Returned

t	Returns t if the cellView argument exists as a cellview in the list of all cells.
nil	Returns nil when the command is not successful.

Example

```
hnlCellInAllCells( cellView )
```

Print Functions

Print functions control netlist formatting and printing.

hnlCompletePrint

```
hnlCompletePrint(
     )
     => t
```

Description

Flushes buffers needed for the `hnlPrintString` function and closes the netlist file.

Arguments

None

Value Returned

`t`	The command is successful.

Example

```
hnlCompletePrint()
```

hnlInitPrint

```
hnlInitPrint(
     t_fileName
     x_lineLength
     t_prefix
     t_postfix
     t_commentStr
     x_softLineLength
     )
     => t / nil
```

Description

Opens the netlist output file specified by filename and sets up the information required by the `hnlPrintString` function.

Arguments

t_fileName	Name of the output file.
x_lineLength	Maximum length of a line in the netlist.
t_prefix	Continuation character placed at the beginning of a continued line.
t_postfix	Continuation character placed at the end of a line to be continued.
t_commentStr	Comment string used by the simulator.
x_softLineLength	Maximum length of a line of output after which folding and continuation of the line need to be considered.

Value Returned

`t`	Returns `t` if the initialization process completes without error.
`nil`	Returns `nil` when the command is not successful.

Example

```
hnlInitPrint( "netlist" 60 "r" "o" "{" 50)
```

hnlPrintString

```
hnlPrintString(
     t_text
     [ g_general ]
     )
     => t / nil
```

Description

Prints the SKILL string text argument to the netlist file.

If a comment is too long to fit on one line, as determined by the hnlMaxLineLength variable, this function converts the comment into multiple single-line comments and adds the comment character, as specified by the hnlCommentStr variable, to the beginning of each line.

If a line is not a comment, but is too long, a new line is inserted after the maximum line length is reached. (The maximum line length is determined by the hnlMaxLineLength variable.) If the hnlLinePostfix variable is not nil, the string specified by this variable is appended to all continued lines. If the hnlLinePrefix variable is not nil, the string specified by this variable is placed at the beginning of all continued lines after the inserted new line.

This function also accepts an optional argument that specifies if a new line should be created after a maximum line length.

Arguments

t_text	Text to be printed to the netlist file.
g_general	This is an optional argument. When set as `t`, it specifies that the text to be printed should not be split when hnlSoftLineLength is reached, even when the text has blank spaces. However, if the number of characters printed on a single line exceeds the length specified by the hnlMaxLineLength variable, the line is broken after printing the string specified by the hnlLinePostFix variable. This argument can be used to print parameter values in the netlist file when parameter values have blank spaces.

Value Returned

`t`	Returns `t` if the given string is printed to the netlist file.
`nil`	Returns `nil` when the command is not successful.

Example

```
hnlPrintString( "The current cell is a netlist primitive." )
```

Netlist TriggerFunctions

Property functions can be used to search for properties and to scale property values.

Miscellaneous Functions

Miscellaneous functions which are used as general utility functions in HNL.

hnlAbortNetlist

```
hnlAbortNetlist(
     )
     => nil
```

Description

Aborts netlisting. When the formatter detects an error during netlisting, it calls this function to inform the netlister to abort netlisting. This function aborts netlisting at the next convenient point, usually after the current cell is processed.

Arguments

None

Value Returned

`nil`	Always returns `nil`.

Example

```
hnlAbortNetlist()
```

hnlDoInstBased

```
hnlDoInstBased(
     l_hnlListOfAllCells
     )
     => t
```

Description

Driver for instance-based netlists. It determines the order in which most of the output functions are called. For each cellview in the schematic hierarchy (determined by the hnlListOfAllCells variable), it sets the hnlCurrentCell global variable to the cellview to be netlisted.

Then, the following global variables are set:

```
hnlCellInputs
hnlCellOutputs
hnlCellOthers
hnlCellOutTerms
hnlCellInTerms
hnlCellOtherTerms
hnlCellNetsOnTerms
```

Next, if this cellview is the top-level cellview (the cellview specified to be netlisted), each of the functions specified in the `hnlTopCellFuncs` list is evaluated in order. If this is not the top-level schematic, each of the functions specified in the `hnlMacroCellFuncs` list is evaluated in order.

Note: None of these functions take arguments.

The current environment is always stored in global variables so that they are available to the output functions. Upon completion, `hnlCurrentCell` is set to `nil`.

Arguments

l_hnlListOfAllCells

List of all cell views to be netlisted. This is a list of lists, and the first element of each sublist is a cell.

```
'( (cell1) (cell2) (cell3) )
```

Value Returned

`t` Always returns `t`.

Example

```
hnlDoInstBased( '( ( cellView1 ) ( cellView2 ) )
```

hnlFindAllCells

```
hnlFindAllCells(
     d_cellView
     l_hnlListOfAllCells
     )
```

Description

Returns a list of all the unique devices and levels of the hierarchy that need to be netlisted. This is a list of lists; the first element of each sublist is a cell.

```
'( (cell1) (cell2) (cell3) )
```

This function recursively traverses the instance hierarchy and adds the master cellviews for each of the devices with the following characteristics to the returned list:

The instance is a “true” instance (i.e., not a terminal).
The master of the instance is not a stopping cellview.

The master does not have a string property `nlAction` set to `ignore`.
The master of the instance contains instances.

Each cellview is added to the `hnlListOfAllCells` list of cells only once, and the list is ordered so the cells that reference other cells occur later in the list than the cells they reference.

Note: Because cellviews can come from multiple libraries, multiple entries may exist with the same `cellName`, but exist in different reference libraries.

Note: In addition, any global net is added to the `hnlAllGlobals` list for later use by the formatting functions.

Note: This function has been optimized for fast database traversal and instance access. Do not override this function unless absolutely necessary because you can slow down the netlisting run time.

Arguments

d_cellView	cellViewId of cell view.
l_hnlListOfAllCells	List of all cell views to be netlisted. This is a list of lists, and the first element of each sublist is a cell. `'( (cell1) (cell2) (cell3) )`

Value Returned

None

hnlIfNoProcedure

```
hnlIfNoProcedure(
     t_progarg
     )
     => value
```

Description

Defines procedures in HNL. By using this function, the netlister only defines a procedure if it is not already defined, thus allowing user-override of netlister functionality by loading user-supplied functions before loading the Cadence netlister.

Note: Source code for this procedure is available for CAD developers in the *install_dir*/tools/dfII/src/hnl/hnl.il file.

Arguments

t_progarg	Body of the procedure.

Value Returned

value	Value returned is dependent on the logic of the procedure.

Example

```
hnlIfNoProcedure( simExecute( cmd )
let( ( status )
simDrain()
status = sh( cmd )
status
)
)
```

hnlPrintDevices

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

Description

Sets up the global variables needed by the output functions to output the connectivity for a device and then calls the `hnlPrintInst` formatter function to print the connectivity for each iteration of each instance in the current cellview being netlisted. The current cellview is determined by the value of the hnlCurrentCell global variable. The global variables set up by this function, which can be used by the formatter function, are as follows:

```
hnlCurrentOutTerms
hnlCurrentInTerms
hnlCurrentOtherTerms
hnlCurrentOutputs
hnlCurrentInputs
hnlCurrentOthers
hnlCurrentType
hnlCurrentInst
hnlCurrentInstName
hnlCurrentIteration
hnlCurrentTermsOnInst
hnlCurrentMaster
```

For a description of these variables, refer to the “Global Variables” section in this chapter.

Note: This function has been optimized for fast database traversal and instance access. Overriding this function will slow down the netlisting process.

Arguments

None

Value Returned

`t`	Returns `t` if no error occurs during processing.
`nil`	Returns nil if the command is not successful.

Example

```
hnlPrintDevices()
```

hnlPrintMessage

```
hnlPrintMessage(
     t_text
     )
     => t
```

Description

Prints the text argument to the stdout port if executed in the Cadence nongraphic environment. If the function is called from within the Cadence graphics environment, the output is written to the CIW window.

You can use this function instead of the following `fprintf` function:

```
fprintf( stdout text )
```

The function is defined in `bin/si` and also in the Cadence graphics program.

You *cannot* modify this function.

Arguments

t_text	Text string.

Value Returned

`t`	Returns `t` upon completion.

Example

```
hnlPrintMessage( "Can't open file simout.tmp\n" )
```

hnlPrintNetlist

```
hnlPrintNetlist(
     l_hnlListOfAllCells
     )
     => t / nil
```

Description

Prints the netlist header, then calls a function to output the connectivity for the netlist, and after that, prints the netlist footer. At this point, all cells are expected to be bound. Error detection for binding is done by the traversal functions.

The netlist is output by calling the following functions in order:

```
hnlPrintNetlistHeader( )
hnlDoInstBased(hnlListOfAllCells)
hnlPrintNetlistFooter( )
```

Arguments

l_hnlListOfAllCells

List of all cell views to be netlisted. This is a list of lists, and the first element of each sublist is a cell.

```
'( (cell1) (cell2) (cell3) )
```

Value Returned

`t`	Returns `t` if no error occurs during processing.
`nil`	Returns `nil` if error occurs during processing.

Example

```
hnlPrintNetlist( '( ( cellView1 ) ( cellView2 ) )
```

Note: Netlisting is controlled by two lists, `hnlListOfAllStopCells` and `hnlListOfAllCells`. To control netlisting of cells such that all the cells of a library are not netlisted, update these two lists before the netlister is invoked, preferably in `hnlPrintNetlistHeader()` or the function called before this.

hnlRunNetlister

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

Description

Entry point for HNL. This function is called to run the hierarchical netlister.

Note: Source code for this procedure is available for CAD developers in the *install_dir*/tools/dfII/src/hnl/hnl.il file.

Arguments

None

Value Returned

t	Returns t if no error is encountered during netlisting.
nil	Returns nil if error is encountered during netlisting.

Example

```
hnlRunNetlister()
```

hnlSetDef

```
hnlSetDef(
     s_sVariable
     g_value
     )
     => t / nil
```

Description

Sets variables in HNL. By using this function, the netlister only sets sVariable if it is not already set, or the symbol sVariable evaluates to null, thus allowing user-override of netlister variables by loading user-supplied defaults before loading the Cadence netlister.

Note: Source code for this procedure is available for CAD developers in the *install_dir*/tools/dfII/src/hnl/hnl.il file.

Arguments

s_sVariable	Symbol name of the variable whose value is set to the *g_value* in the second argument. The value is set only if it is not already set, or if the symbol sVariable evaluates to null.
g_value	Value assigned to sVariable if it is null.

Value Returned

t	Returns t if the variable is set to the value given.
nil	Returns nil if the variable is not set to the value given.

Example

```
hnlSetDef( 'simSimulator "spice" )
```

hnlSetPrintLinePrefix

```
hnlSetPrintLinePrefix(
     t_value
     )
     => t / nil
```

Description

Registers the line prefix to be used when printing individual subcircuits during auCdl netlisting to indicate the continuation of the text on the next line, when the line length exceeds the limit set by the HNL global variable `hnlSoftLineLength`.

Arguments

t_value	The line prefix to use as the indicator of the continuation of subcircuit text. The valid prefixes are: ■ `+` ■ `*.PININFO`

Value Returned

`t`	Returns `t` if the command is successful.
`nil`	Returns `nil` if the command is not successful.

Example

```
hnlSetPrintLinePrefix("+")
```

hnlGetPrintLinePrefix

```
hnlGetPrintLinePrefix(
     )
     => t_value
```

Description

Returns the line prefix set using hnlSetPrintLinePrefix, which is used to print individual subcircuits during auCdl netlisting to indicate the continuation of the text on the next line, when the line exceeds the maximum character limit.

Arguments

None

Value Returned

t_value	The prefix set as the indicator of the continuation of the subcircuit text in auCdl netlists. The prefix can be one of the following: ■ + ■ *.PININFO

Example

```
hnlGetPrintLinePrefix()
```

hnlSetPseudoTermDir

```
hnlSetPseudoTermDir(
     t_string
     )
     => t / nil
```

Description

Sets the direction of pseudo ports, which are created to propagate inherited connections from a cellview to upper levels in the hierarchy. OSS-based formatter calls this function to set the direction of pseudo ports. If the formatter does not call it, you can specify a statement to call the function in the `.simrc` file.

Arguments

t_string	Direction of pseudo ports. The argument can have any of these values: ■ `input`: Indicates that the direction of pseudo ports is input. ■ `output`: Indicates that the direction of pseudo ports is output. ■ `inputOutput`: Indicates that the direction of pseudo ports is `inputOutput`. ■ `sameAsTermDir`: Indicates that the direction of pseudo port created for explicit inherited terminals, is same as explicit terminal. The direction of rest of the pseudo ports is `inputOutput`.

Value Returned

`t`	Returns `t` if no error is encountered during netlisting.
`nil`	Returns `nil` if error is encountered during netlisting.

Example

```
hnlSetPseudoTermDir("inputOutput")
```

hnlSetVars

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

Description

Sets all global-required variables for netlisting. Depending on the target simulator, the required netlist formatting functions are loaded.

After that, the `hnlViewList` and `hnlStopList` global variables are set to be lists of strings from the user-entered values for the view switch list and the stopping view list. The last step is to load a user-supplied file of functions if specified. If netlisting is not to continue, this function returns `nil`. If there is an error, the `hnlError` variable is also set to `true`. When run from within SE, this function sets the appropriate `hnl` variables from ones in the SE environment.

Note: Source code for this procedure is available for CAD developers in the *`install_dir`*`/tools/dfII/src/hnl/hnl.il` file.

Arguments

None

Value Returned

`t`	Returns `t` if no error is encountered during processing.
`nil`	Returns `nil` if error is encountered during processing.

Example

```
hnlSetVars()
```

hnlSortTerms

```
hnlSortTerms(
     d_term1
     d_term2
     )
     => t / nil
```

Description

Determines net order in the sorted lists of net names. It takes as argument two terminals and does a comparison by the name field. It is used as argument to the SKILL `sort` function.

Arguments

d_term1	termId of terminal 1.
d_term2	termId of terminal 2.

Value Returned

`t`	Returns `t` if the name of the net connected to `term1` precedes that of the net connected to `term2`.
`nil`	Returns `nil` if the name of the net connected to `term1` does not precede that of the net connected to `term2`.

Example

```
hnlSortTerms( term1 term2 )
```

hnlSortTermsToNets

```
hnlSortTermsToNets(
     l_terminals
     )
     => list
```

Description

Takes as argument a list of terminals and returns a list of lists. The first element in each list is the original terminal, the second element is the name of the net attached to that terminal. The list is sorted alphanumerically by the name of the terminal attached to the net.

The hnlCurrentIteration global variable is used to construct the name of the net attached to the current iteration of the current instance being expanded. For this reason, this function can only be called from the output-formatting functions for a stopping cell.

For inherited terminals, the *netlister-generated* names of the signals attached to the terminal are returned.

Arguments

l_terminals	List of terminal identifications (termId) of terminals.

Value Returned

list	Returns a list of lists. The first element in each list is the original terminal, and the second element is the name of the net attached to that terminal. The list is sorted alphanumerically by the name of the terminal attached to the net.

Example

```
hnlSortTermsToNets( '( term1 term2 term3 term4 term5 ) )
```

hnlStartNetlist

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

Description

Opens the output files.

Note: Source code for this procedure is available for CAD developers in the *install_dir*/tools/dfII/src/hnl/hnl.il file.

Arguments

None

Value Returned

t	Returns t if no error is encountered during netlisting.
nil	Returns nil if error is encountered during netlisting.

Example

```
hnlStartNetlist()
```

hnlStopNetlist

```
hnlStopNetlist(
     )
     => t
```

Description

Cleans up after netlisting, closes open files, closes all open cellviews, and resets all global variables to nil.

Note: Source code for this procedure is available for CAD developers in the *install_dir*/tools/dfII/src/hnl/hnl.il file.

Arguments

None

Value Returned

t	Always returns t.

Example

```
hnlStopNetlist()
```

hnlStringToList

```
hnlStringToList(
     t_theString
     )
     => list
```

Description

Given a string of white-space-separated strings, this function returns a list of strings, that is, the string - silos schematic - returns the list (silos schematic).

Note: Source code for this procedure is available for CAD developers in the *install_dir*/tools/dfII/src/hnl/hnl.il file.

Arguments

t_theString	A string of white-space-separated strings.

Value Returned

list	A list of strings.

Example

```
hnlStringToList( "behavorial structural verilog schematic symbol" )
```

Name-Mapping Functions

Name-mapping functions implement the name-mapping feature of HNL. You can use the hnlMapName, hnlMapNetName, hnlMapTermName, hnlMapInstName, and hnlMapModelName functions during netlisting. The initialization and write functions are called by the HNL driver as part of the netlisting process.

hnlGetMappedInstNames

```
hnlGetMappedInstNames(
     )
     => list
```

Description

Returns the list of all names mapped using the `hnlMapInstName` function. This function can only be called during netlisting.

Arguments

None

Value Returned

list	Returns the list of all names mapped using the `hnlMapInstName()` function.

Example

```
hnlGetMappedInstNames()
```

hnlGetMappedModelNames

```
hnlGetMappedModelNames(
     )
     => list
```

Description

Returns the list of all names mapped using the `hnlMapModelName` function. This function can only be called during netlisting.

Arguments

None

Value Returned

list	Returns the list of all names mapped using the `hnlMapModelName()` function.

Example

```
hnlGetMappedModelNames()
```

hnlGetMappedNames

```
hnlGetMappedNames(
     )
     => list
```

Description

Returns the list of all names mapped using the `hnlMapName` function. This function can only be called during netlisting.

In the case of inherited terminals and inherited connections, the *netlister-generated* names are returned.

Arguments

None

Value Returned

list	Returns the list of all names mapped using the `hnlMapName()` function.

Example

```
hnlGetMappedNames()
```

Assuming that *hnlMapName()* is used to map all names in the entire design, the value is list(“inh_gnd” “inh_vdd”). These two are mapped names for the local nets *vdd!* and *gnd!* in the *schematic* view of the cell *inv*.

hnlGetMappedNetNames

```
hnlGetMappedNetNames(
     )
     => list
```

Description

Returns the list of all names mapped using the `hnlMapNetName` function. This function can only be called during netlisting.

In the case of inherited terminals and inherited connections, the *netlister-generated* names are returned.

Arguments

None

Value Returned

list	Returns the list of all names mapped using the `hnlMapNetName()` function.

Example

```
hnlGetMappedNetNames()
```

Assuming that *hnlMapNetName()* are used to map all net names in the entire design, the value is list(“inh_gnd” “inh_vdd”). These two are mapped names for the local nets *vdd!* and *gnd!* in the *schematic* view of the cell *inv.*

hnlInitMap

```
hnlInitMap(
     x_maxNameLength
     x_maxInstNameLength
     x_maxModelNameLength
     x_maxNetNameLength
     x_maxTermNameLength
     t_namePrefix
     t_netNamePrefix
     t_instNamePrefix
     t_modelNamePrefix
     t_termNamePrefix
     t_hierarchyDelimeter
     l_invalidFirstChars
     l_invalidCharsInName
     l_hnlMapNetFirstChar
     l_hnlMapNetInName
     l_hnlMapInstFirstChar
     l_hnlMapInstInName
     l_hnlMapModelFirstChar
     l_hnlMapModelInName
     l_hnlMapTermFirstChar
     l_hnlMapTermInName
     t_mapName
     t_designName
     t_viewList
     t_stopList
     l_hnlInvalidNames
     l_hnlInvalidNetNames
     l_hnlInvalidInstNames
     l_hnlInvalidModelNames
     l_hnlInvalidTermNames
     )
     => t / nil
```

Description

Initializes the variables needed for name translation.

Arguments

x_maxNameLength	Maximum number of characters that can be in a name.
x_maxInstNameLength	Maximum number of characters that can be in an instance name. If not given, maxNameLength is used as the default.
x_maxModelNameLength	Maximum number of characters that can be in a model name. If not given, maxNameLength is used as the default.
x_maxNetNameLength	Maximum number of characters that can be in a net name. If not given, maxNameLength is used as the default.
x_maxTermNameLength	Maximum number of characters that can be in a terminal name. If not given, maxNameLength is used as the default.
t_namePrefix	String prefix used when creating new names using the `hnlMapName()` function. If this argument is “hnl_,” subsequent calls to the `hnlMapName` function return names such as “hnl_9” when a name requires full mapping.
t_netNamePrefix	String prefix used when creating new names using the `hnlMapNetName()` function. If this argument is “net,” subsequent calls to the function `hnlMapNetName` return names such as “net9” when a name requires full mapping.
t_instNamePrefix	String prefix used when creating new names using the `hnlMapInstName( )` function. If this argument is “inst,” subsequent calls to the `hnlMapInstName()` function return names such as “inst9” when a name requires full mapping.
t_modelNamePrefix	String prefix that should be used when creating new names using the `hnlMapModelName()` function. If this argument is “model,” subsequent calls to the `hnlMapModelName()` function return names such as “model9” when a name requires full mapping.

t_termNamePrefix	String prefix that should be used when creating new names using the `hnlMapTermName()` function. If this argument is `term`, subsequent calls to the `hnlMapTermName` function return names such as `term19` when a name requires full mapping.
t_hierarchyDelimeter	Hierarchy delimiter that the target simulator uses when creating flat names from the hierarchical netlist. For SILOS, this is "(". This argument should be a SKILL string containing a single character. This argument is later used when translating full pathnames from the simulator output.
l_invalidFirstChars	SKILL list that specifies which characters may not begin a name for the target simulator when using the `hnlMapName( )` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name beginning with this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character if it is encountered. This list is used when mapping names with the `hnlMapName()` function.
l_invalidCharsInName	SKILL list that specifies which characters may not be contained in a name for the target simulator when using the `hnlMapName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name containing this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character when it is encountered. This list is used when mapping names with the h`nlMapName()` function.

l_hnlMapNetFirstChar

SKILL list that specifies which characters may not begin a name for the target simulator when using the `hnlMapNetName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name beginning with this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character if it is encountered. This list is used when mapping names with the `hnlMapNetName()` function.

l_hnlMapNetInName

SKILL list that specifies which characters may not be contained in a name for the target simulator when using the `hnlMapNetName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name containing this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character when it is encountered. This list is used when mapping names with the `hnlMapNetName()` function.

l_hnlMapInstFirstChar — SKILL list that specifies which characters may not begin a name for the target simulator when using the `hnlMapInstName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name beginning with this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character if it is encountered. This list is used when mapping names with the `hnlMapInstName()` function.

l_hnlMapInstInName — SKILL list that specifies which characters may not be contained in a name for the target simulator when using the `hnlMapInstName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name containing this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character when it is encountered. This list is used when mapping names with the `hnlMapInstName()` function.

l_hnlMapModelFirstChar SKILL list that specifies which characters may not begin a name for the target simulator when using the `hnlMapModelName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name beginning with this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character if it is encountered. This list is used when mapping names with the h`nlMapModelName()` function.

l_hnlMapModelInName SKILL list that specifies which characters may not be contained in a name for the target simulator when using the `hnlMapModelName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name containing this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character when it is encountered. This list is used when mapping names with the `hnlMapModelName( )` function.

l_hnlMapTermFirstChar	SKILL list that specifies which characters may not begin a name for the target simulator when using the `hnlMapTermName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name beginning with this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character if it is encountered. This list is used when mapping names with the `hnlMapTermName()` function.
l_hnlMapTermInName	SKILL list that specifies which characters may not be contained in a name for the target simulator when using the `hnlMapTermName()` function. Each element in the list may be either a string or another list. If it is a string, it should contain a single character. This specifies that there is no replacement for this character and any name containing this character must be replaced. If an element is another list, it should contain two elements. The first element should be a string containing a single character that is the invalid character. If the second element of the sublist does not exist or is `nil`, the character specified by the first element is removed. If the second element is a string, that string replaces the character when it is encountered. This list is used when mapping names with the `hnlMapTermName()` function.
t_mapName	Name of the file in which the name map is stored.
t_designName	Full name of the top level design. Example: `/mnt/dave/alu/schematic/current.`
t_viewList	View switch list that is used for netlisting. Must be a SKILL string type. Example: `"silos schematic"`
t_stopList	Stopping points used for netlisting. Must be a SKILL string type. Example: `"silos mysilos"`

l_hnlInvalidNames	A list of names which are invalid in the target tool name space. This is associated with “generic” names. This should not be used if you are using the following four variables. They are mutually exclusive. Example: `'("begin" "end" "for")`
l_hnlInvalidNetNames	A list of names which are invalid as net names in the target tool net name space. Example: `'("begin" "end" "for" "net")`
l_hnlInvalidInstNames	A list of names which are invalid as instance names in the target tool instance name space. Example: `'("begin" "end" "for" "instance")`
l_hnlInvalidModelNames	A list of names which are invalid as model names in the target tool model name space. Example: `'("begin" "end" "for" "model")`
l_hnlInvalidTermNames	A list of names which are invalid as terminal names in the target tool terminal name space. Example: `'("begin" "end" "for" "terminal")`

Value Returned

`t`	The `hnlInitMap` function returns `t` if successful.
`nil`	The `hnlInitMap` function returns `nil` if it is not successful.

Example

```
unless( hnlInitMap( hnlMaxNameLength hnlNamePrefix
       maxInstNameLength maxModelNameLength
      maxNetNameLength maxTermNameLength
      simNetNamePrefix simInstNamePrefix
      simModelNamePrefix simTermNamePrefix
      hnlHierarchyDelimeter hnlMapIfFirstChar
      hnlMapIfInName hnlMapNetFirstChar
      hnlMapNetInName hnlMapInstFirstChar
      hnlMapInstInName hnlMapModelFirstChar
      hnlMapModelInName hnlMapTermFirstChar
      hnlMapTermInName hnlMapFileName
      FullMasterName viewListString
      stopListString hnlInvalidNames
      hnlInvalidNetNames hnlInvalidInstNames
      hnlInvalidModelName hnlInvalidTermNames )
   sprintf(errorMessage
      "Netlister: Can't initialize name mapping functions.")
   println( errorMessage )
```

```
   return( nil )
)
```

hnlMapInstName

```
hnlMapInstName(
     t_cdsName
     t_prefix
     )
     => name
```

Description

Returns a new mapped name. The `cdsName` string is a mandatory parameter and the prefix string is an optional one. If the argument specified in *`cdsName`* is legal, the same name is returned. Otherwise, OSS searches for the prefix. In case, the prefix is legal, then the value of global counter, which starts with 0, is appended to the prefix. If the prefix is not legal or `null`, a new name using the global instance prefix and the global counter is generated. However, if both the prefix and the global instance prefix are illegal or `null`, then an error message is displayed.

Arguments

`t_cdsName`	Name of an instance.
`t_prefix`	Name of an instance prefix.

Value Returned

`name`	Returns a new mapped name if the argument specified as the *`cdsName`* is not a valid name for the target simulator. Otherwise, the string is returned or an error message is displayed.

Example

```
hnlMapInstName( "inst1" )
```

hnlMapModelName

```
hnlMapModelName(
     t_cdsName
     [ t_viewList
     t_pathName
     t_LibName
     t_cellName
     t_viewName
     t_isCellTopCell ]
     )
     => name
```

Description

Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the string is returned. The `hnlMapModelFirstChar`, `hnlMapModelInName`, `hnlMaxNameLength`, and `simModelNamePrefix` variables are used to determine the mapping of the name.

Arguments

t_cdsName	Name of a model.
	If Hierarchical Configuration is used, the following arguments also should be supplied:

If Hierarchical Configuration is used, the following arguments also should be supplied:

t_viewList	Effective view list
t_pathName	Path name string
t_LibName	Configuration library name
t_cellName	Cell name
t_viewName	View name
t_isCellTopCell	If this cell is top cell

Value Returned

name	Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the string is returned.

Example

```
hnlMapModelName( "block1" )
```

hnlMapName

```
hnlMapName(
      t_cdsName
      )
      => name
```

Description

Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the `cdsName` string is returned.

The `hnlMapIfFirstChar`, `hnlMapIfInName`, `hnlNamePrefix`, and `hnlMaxNameLength` variables are used to determine if a name is invalid. If an alternate string is specified for a invalid character, the string replaces the invalid character to create a new name. Characters can also be deleted from a name. If the name is too long or there is no character map, a new name is generated by using the prefix specified by the `hnlNamePrefix` variable, and suffixing it with a unique number. Assume the variables are set as in the following list:

```
hnlNamePrefix = "hnl_"
hnlMaxNameLength = 10
hnlMapIfFirstChar = '(("@" "__") "0" "1")
hnlMapIfInName = '(("@" "__"))
```

Then, `hnlMapName`("alu@99") would return "alu__99" and `hnlMapName`("0.Y") would return something like "hnl_12." For more information on the mapping abilities, refer to the `hnlMapIfInName` and `hnlMapIfFirstChar` variables, as well as the `hnlInitMap()` function.

Note: If you use this function, you cannot use the `hnlMapInstName`, `hnlMapModelName`, `hnlMapTermName`, and `hnlMapNetName` functions.

In the case of inherited terminals and inherited connections, *netlister-generated* names are returned. `hnlMapName()` cannot be used interchangeably with `hnlMapNetName()` or other type-specific name mapping functions in the same netlisting session.

Arguments

t_cdsName	String name to be mapped.

Value Returned

name	Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the `cdsName` string is returned.

Example

```
hnlMapName( "alu@99" )
```

For net *vdd!* in the *schematic* view of cell *inv*, *hnlMapName(“vdd!”)* returns “inh_vdd”. For net *gnd!* in the same cellview, *hnlMapName(“gnd!”)* returns “inh_gnd”. These two are *netlister-generated* names due to inherited connection.

hnlMapNetName

```
hnlMapNetName(
     t_cdsName
     )
     => name
```

Description

Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the `cdsName` string is returned.

The `hnlMapNetFirstChar`, `hnlMapNetInName`, `hnlMaxNameLength`, and `simNetNamePrefix` variables are used to determine the mapping of the name.

In the case of inherited terminals and inherited connections, *netlister-generated* names are returned. `hnlMapNetName()` or other type-specific name mapping functions cannot be used interchangeably with `hnlMapName()` in the same netlisting session.

Arguments

t_cdsName	Name of a net.

Value Returned

name	Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the `cdsName` string is returned.

Example

```
hnlMapNetName( "netA" )
```

For net *vdd!* in the *schematic* view of cell *inv*, *hnlMapName(“vdd!”)* returns “inh_vdd”. For net *gnd!* in the same cellview, *hnlMapName(“gnd!”)* returns “inh_gnd”. These two are *netlister-generated* names due to inherited connection.

hnlMapTermName

```
hnlMapTermName(
     t_cdsName
     )
     => name / nil
```

Description

Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the `cdsName` string is returned.

The `hnlMapTermFirstChar`, `hnlMapTermInName`, `hnlMaxNameLength`, and `simTermNamePrefix` variables are used to determine the mapping of the name.

Arguments

t_cdsName	Name of a terminal.

Value Returned

name	Returns a new mapped name if the argument specified as the `cdsName` string is not a valid name for the target simulator. Otherwise, the `cdsName` string is returned.
`nil`	Returns nil if the command fails.

Example

```
hnlMapTermName( "IN1" )
```

hnlWriteMap

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

Description

Stores the name map tables in a file specified in an earlier call to `hnlInitMap` function. This function takes no arguments.

If the function was successful, `t` is returned; if the function was unsuccessful, an error message is printed and `nil` is returned.

Note: The syntax of the map file cannot be relied upon because it is subject to change without notice. Always use Cadence-supplied access functions to translate names.

Arguments

None

Value Returned

`t`	Returns `t` if no error is encountered during processing.
`nil`	Returns `nil` if error is encountered during processing.

Example

```
hnlWriteMap()
```

With the Incremental Hierarchical Netlister (IHNL) you can design a formatter that netlists incrementally. An incremental formatter checks each cellview in the design and netlists only the cellviews that the designer has modified since the previous netlisting. IHNL is only an option to HNL; it is not a separate netlister. It is important that you read the basic HNL documentation first, before reading this section, which describes only additional functionality to allow your formatter to be used in the incremental netlisting mode. For simplicity, IHNL will be referred to as the hierarchical netlister running with the incremental feature.

hnlNmpSetNameSpaces

```
hnlNmpSetNameSpaces(
     t_source
     t_dest
     )
     => t / nil
```

Description

Specifies the nmp namespaces to perform namespace mapping during OSS netlisting. The function accepts Cadence nmp supported namespaces as arguments. You can run the `%> nmp getSpaceNames` command to display a list of valid namespaces.

Arguments

t_source	Source namespace from which mapping is to be done usually `CDBA`.
t_dest	Namespace in which netlist is to be written.

Values Returned

`t`	Returns `t` if the function executes successfully.
`nil`	Returns `nil` if the function does not execute successfully.

Example

```
hnlNmpSetNameSpaces("CDBA" "VHDL")
```

hnlSetMappingType

```
hnlSetMappingType(
     t_string
     )
     => t / nil
```

Description

Sets namespace mapping type for nets, terminals, globals, and models to one of the following: tabularOnly, nmpOnly, nmpWithTabular.

- tabularOnly: Is a default OSS name mapping type.
- nmpOnly: Maps invalid names in a formatter name space. The function returns `null`, if nmp fails.
- nmpWithTabular: Maps invalid names in a formatter name space similar to `nmpOnly` mapping type. However, if nmp fails or there is a collision with already mapped names, OSS reverts back to `tabularOnly` maping type.

Cadence and third party formatters, which are plugged into OSS, use the `hnlSetMappingType()` function.

Arguments

t_string	Mapping type.

Values Returned

`t`	Returns `t` if the function executes successfully.
`nil`	Returns `nil` if the function does not execute successfully.

Examples

```
hnlSetMappingType("tabularOnly")
hnlSetMappingtype("nmpOnly")
hnlSetMappingType("nmpWithTabular")
```

hnlIsCVInUserStopCVList

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

Description

Determines whether a cellview, which is present in `hnlUerStopCVList`, is considered a primitive or a stop view while netlisting.

Arguments

d_cellViewId	Id of the cellview, which is present in `hnlUerStopCVList`.

Value Returned

`t`	Returns `t` if the function executes successfully.
`nil`	Returns `nil` if the function does not execute successfully.

Example

```
hnlIsCVInUserStopCVList( hnlCurrentMaster )
```

Functions for Incremental Netlisting

Use the following functions when you write an incremental netlist formatter.

hnlCloseCellFiles

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

Description

Closes all files opened to netlist `hnlCurrentCell`. This function calls the `hnlWriteBlockControlFile` function to create the `control` file under the directory for `hnlCurrentCell`.

Arguments

None

Value Returned

`t`	Returns `t` if no error is encountered during processing.
`nil`	Returns `nil` if an error is encountered during processing.

Example

```
hnlCloseCellFiles()
```

hnlGenIncludeFile

```
hnlGenIncludeFile(
     )
     => value
```

Description

Creates the `include` file. You must define this function for a formatter that needs an `include` file. Use the `hnlIncludeFileName` variable to specify the name of the `include` file. Use the `hnlIncludeFile` variable to access the file pointer. IHNL calls the `hnlGenIncludeFile` function after it generates the netlist files.

By default this function does not create an `include` file.

Arguments

None

Value Returned

value	Value returned depends on the logic of your function.

Example

```
hnlIfNoProcedure( hnlGenIncludeFile()
   let( ( name )
      if( hnlIncrementalMode
      then t
      else if( hnlAllGlobals
         then fprintf( hnlIncludeFile ".GLOBAL" )
            foreach( name hnlAllGlobals
               fprintf( hnlIncludeFile " %s"
                     hnlGetGlobalNetMappedName( name ) )
            )
            fprintf( hnlIncludeFile "\n" )
            t
         else hnlCatIncrementalNetlistFiles( hnlIncludeFile,
                     hnlNetlistFileList )
         )
      )
```

```
    )
)
```

hnlCatIncrementalNetlistFiles()

```
hnlCatIncrementalNetlistFiles(
     p_catFile
     l_netlistFileList
     )
     => t / nil
```

Description

This function concatenates the list of netlist files into a single file whose file handle is given as the first parameter to this function. The second parameter is the list of files to be concatenated.

Arguments

p_catFile	File handle of the output file
l_netlistFileList	Names of the netlist files to be concatenated.

Value Returned

t	The command is successful.
nil	The command is not successful.

Example

```
hnlCatIncrementalNetlistFiles( hnlIncludeFile,
                     hnlNetlistFileList )
```

hnlGetGlobalModelMappedName

```
hnlGetGlobalModelMappedName(
     t_cellName
     t_switchList
     )
     => name
```

Description

Given the name of cell name, this function returns a new mapped name if the argument specified as the cellName string is not a valid name for the target tool. Otherwise, an empty string is returned.

Arguments

t_cellName	Name of the global cell.
t_switchList	If a configuration is used, this argument is the switch list, else this argument is not used

Value Returned

name	Returns the mapped name of the given global cell.

Example

```
hnlGetGlobalModelMappedName( "nand" "schematic spice" )
```

hnlGetGlobalNetMappedName

```
hnlGetGlobalNetMappedName(
     t_netName
     )
     => name
```

Description

Given the name of a global net, this function returns a new mapped name if the argument specified as the netName string is not a valid name for the target tool. Otherwise, an empty string is returned.

Arguments

t_netName	Name of global net.

Value Returned

name	Returns the mapped names of a given global net.

Example

```
hnlGetGlobalNetMappedName( "GND!" )
```

hnlMakeNetlistFileName

```
hnlMakeNetlistFileName(
     d_cellViewId
     t_effectiveViewList
     t_currentCellPath
     t_configLibName
     t_configCellName
     t_configViewName
     g_isTopConfigCell
     )
     => name
```

Description

Returns the relative name of the netlist file for `cellViewId`. This relative name is the path in the current netlisting directory. The `cellViewId` must be a nonstopping cellview, which you can netlist.

Arguments

d_cellViewId	cellViewId of cell view for which netlist file name is to be created.
t_effectiveViewList	Effective view list.
t_currentCellPath	Path to the current cell.
t_configLibName	Library name for effective config.
t_configCellName	Cell name for effective config for this cell view
t_configViewName	View name for effective config for this cell view.
g_isTopConfigCell	If the cellview is top cell.

Value Returned

name	Returns the relative name of the netlist file for `cellViewId`.

Examples

The following example shows how to call the `hnlMakeNetlistFileName()` function when HDB configuration is used for netlisting:

```
hnlMakeNetlistFileName( hnlCurrentCell hnlCurrentCellViewList hnlCurrentCellPath
hnlCurrentCellConfigLibname hnlCurrentCellConfigCellname
hnlCurrentCellConfigViewname hnlIsTopConfigCell  )
```

The variables, in the example, are replaced with their values at run-time.

The following example shows how to call the `hnlMakeNetlistFileName()` function when HDB configuration is not used for netlisting:

```
hnlMakeNetlistFileName( cellView )
```

hnlMapCellName

```
hnlMapCellName(
     d_cellViewId
     [ t_viewList
     t_pathName
     t_LibName
     t_cellName
     t_viewName
     t_isCellTopCell ]
     )
     => name
```

Description

Returns the netlist module name of the cellview (cellViewId). It adds the <cell name><module name> mapped pair into the model section of the name map associated with the hnlCurrentCell variable. The cellViewId must be a non-stopping instance.

Arguments

d_cellViewId	CellViewId of cell view.

If Hierarchical Configuration is used, the following arguments also should be supplied:

t_viewList	Effective view list
t_pathName	Path name string
t_LibName	Configuration library name
t_cellName	Cell name
t_viewName	View name
t_isCellTopCell	If this cell is top cell

Value Returned

name	Returns the netlist module name of the cellview (cellViewId).

Example

```
hnlMapCellName( cvId  "schematic spice" "/I1/I3"  "myLib" "middle"
"schematic"  FALSE )
```

hnlMapCellModuleName

```
hnlMapCellModuleName(
     d_cellViewId
     [ t_viewList
     t_pathName
     t_LibName
     t_cellName
     t_viewName
     t_isCellTopCell ]
     )
     => name
```

Description

Returns the netlist module name of the cellview (*cellViewId*). It adds the <cell name><module name> mapped pair into the model section of the name map associated with the hnlCurrentCell variable. The *cellViewId* must be a nonstopping instance. For a design opened in the configuration view, you must specify the additional arguments.

Arguments

d_cellViewId	cellViewId of cell view.

If Hierarchical Configuration is used, supply the following arguments:

t_viewList	Effective view list
t_pathName	Path name string
t_LibName	Configuration library name
t_cellName	Cell name
t_viewName	View name
t_isCellTopCell	If this cell is top cell

Value Returned

Returns the netlist module name of the cellview (cellViewId).

Examples

Example 1: You can generate a SILOS netlist in which an instance in a schematic called 122 appears as follows:

```
(INST1 CKT122
```

Use the following code in your formatter:

```
fprintf( hnlNetlistFile
         hnlMapName( hnlCurrentInstName) )
fprintf( hnlNetlistFile
         hnlMapCellModuleName( hnlCurrentMaster ) )
```

To create the netlist for the schematic instance 122, do the following:

```
fprintf( hnlNetlistFile "\n.MACRO %s"
         hnlMapCellModuleName( hnlCurrentCell ) )
```

Example 2: When working with the configuration view of a design, you can use the `hnlMapCellModuleName` function for the cell header in the `hnlPrintTopCellHeader` or `hnlPrintDeviceHeader` functions as follows:

```
mapname = hnlMapCellModuleName(
                    hnlCurrentCell
                    hnlCurrentCellViewList
                    hnlCurrentCellPath
                    hnlCurrentCellConfigLibname
                    hnlCurrentCellConfigCellname
                    hnlCurrentCellConfigViewname
                    hnlIsTopConfigCell
                    )
```

In the configuration view, you can also use the `hnlMapCellModuleName` function for a non-stopping cell in the `hnlPrintInst` function as follows:

```
mapname = hnlMapCellModuleName(
                    hnlCurrentMaster
                    hnlCurrentMasterViewList
                    hnlCurrentMasterPathString
                    hnlCurrentMasterConfigLibName
                    hnlCurrentMasterConfigCellName
                    hnlCurrentMasterConfigViewName
                    hnlCurrentMasterIsTopCellConfig
                    )
```

hnlSetCellFiles

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

Description

Opens all the files associated with `hnlCurrentCell` and re initializes name mapping functions.

Arguments

None

Value Returned

`t`	Returns `t` if no error is encountered during processing.
`nil`	Returns `nil` if error is encountered during processing.

Example

```
hnlSetCellFiles()
```

hnlWriteBlockControlFile

```
hnlWriteBlockControlFile(
     d_cellView
     [ t_viewList
     t_pathName
     t_LibName
     t_cellName
     t_viewName
     t_isCellTopCell ]
     )
     => t / nil
```

Description

Creates the control file for cellView. The control file records information for subsequent IHNL runs. The hnlCloseCellFiles function calls the hnlWriteBlockControlFile function.

Arguments

d_cellView	cellViewId of cell view.

If Hierarchical Configuration is used, supply the following arguments:

t_viewList	Effective view list
t_pathName	Path name string
t_LibName	Configuration library name
t_cellName	Cell name
t_viewName	View name
t_isCellTopCell	If this cell is top cell

Value Returned

t	Returns t if no error is encountered during processing.
nil	Returns nil if error is encountered during processing.

Example

```
hnlWriteBlockControlFile( cellView )
```

HNL Trigger Functions for Pre and Post Netlist Customization

The following HNL functions are available for customizing the netlist as per user requirements after the netlisting is done by the tool:

hnlRegPreNetlistTrigger

```
hnlRegPreNetlistTrigger(
     S_triggerFunc
     )
     => t / nil
```

Description

Registers a trigger, which is a user-defined SKILL procedure that is called before the netlist generation.

The registration can be done in the `.simrc` file, at the CIW, or any other location that will be executed before generating the netlist.

Arguments

S_triggerFunc	A symbol representing a user-defined function that needs to be called before netlist generation.

Value Returned

`t`	Returns `t` if the trigger registration completed successfully.
`nil`	Returns `nil` if the trigger registration could not be completed.

Example

User-defined SKILL procedure, which will be called before netlist generation:

```
procedure( Func()
   let(()
      printf("In Func\n")
   )
)
```

If you want the above procedure `Func` to be run only for Verilog and SytemVerilog netlisting, set `simSimulator=="verilog"`. For Spectre, set `simSimulator=="spectre"`. If you do not set this variable, then the trigger will run for all netlisters.

Note: If more than one triggers have been registered, then they are run in the order of registration.

```
when(simSimulator == "verilog"
   hnlRegPreNetlistTrigger('Func)
)
```

hnlDeRegPreNetlistTrigger

```
hnlDeRegPreNetlistTrigger(
     S_triggerFunc
     )
     => t / nil
```

Description

Deregisters a trigger, which is a user-defined SKILL procedure that has been registered using `hnlRegPreNetlistTrigger`.

Arguments

S_triggerFunc	A symbol representing a user-defined function that needs to be deregistered.

Value Returned

`t`	Returns `t` if the trigger was deregistered successfully.
`nil`	Returns `nil` if the trigger deregistration was not completed successfully.

Example

To deregister the user-defined SKILL procedure `Func`, which was registered through `hnlRegPreNetlistTrigger`.

```
procedure( Func()
   let(()
      printf("In Func\n")
   )
)

when(simSimulator == "verilog"
   hnlRegPreNetlistTrigger('Func)
)

hnlDeRegPreNetlistTrigger('Func)
=> t
```

hnlPreNetlistTriggerList

```
hnlPreNetlistTriggerList(

     )
     => l_triggerFunc / nil
```

Description

Returns the list of functions registered through `hnlRegPreNetlistTrigger`.

Arguments

None

Value Returned

l_triggerFunc	Returns a list of functions registered through `hnlRegPreNetlistTrigger`.
`nil`	Returns `nil` if no function was registered.

Example

To view the list of functions registered through `hnlRegPreNetlistTrigger`:

```
hnlPreNetlistTriggerList()
=> nil
```

hnlRegPostNetlistTrigger

```
hnlRegPostNetlistTrigger(
     S_triggerFunc
     )
     => t / nil
```

Description

Registers a trigger, which is a user-defined SKILL procedure that is called after netlist generation.

The registration can be done in the `.simrc` file, at the CIW, or any other location that will be executed before generating the netlist.

Arguments

S_triggerFunc	A symbol representing a user-defined function that needs to be called after netlist generation.

Value Returned

`t`	Returns `t` if the trigger registration completed successfully.
`nil`	Returns `nil` if the trigger registration could not be completed.

Example

User-defined SKILL procedure, which will be called after netlist generation:

```
procedure( Func()
   let(()
      printf("In Func\n")
   )
)
```

If you want the above procedure `Func` to be run only for Verilog and SytemVerilog netlisting, set `simSimulator=="verilog"`. For Spectre, set `simSimulator=="spectre"`. If you do not set this variable, then the trigger will run for all netlisters.

Note: If more than one triggers have been registered, then they are run in the order of registration.

```
when(simSimulator == "verilog"
   hnlRegPostNetlistTrigger('Func)
)
```

hnlPostNetlistTriggerList

```
hnlPostNetlistTriggerList(

     )
     => l_triggerFunc / nil
```

Description

Returns the list of functions registered through `hnlRegPostNetlistTrigger`.

Arguments

None

Value Returned

l_triggerFunc	List of functions registered through `hnlRegPostNetlistTrigger`.
`nil`	Returns `nil` if no function was registered.

Example

To view the list of functions registered through `hnlRegPostNetlistTrigger`:

```
hnlPostNetlistTriggerList()
=> nil
```

hnlDeRegPostNetlistTrigger

```
hnlDeRegPostNetlistTrigger(
     S_triggerFunc
     )
     => t / nil
```

Description

Deregisters a trigger, which is a user-defined SKILL procedure that has been registered using `hnlRegPostNetlistTrigger`.

Arguments

S_triggerFunc	A symbol representing a user-defined function that needs to be deregistered.

Value Returned

`t`	Returns `t` if the trigger was deregistered successfully.
`nil`	Returns `nil` if the trigger deregistration was not completed successfully.

Example

To deregister the user-defined SKILL procedure `Func`, which was registered through `hnlRegPostNetlistTrigger`.

```
procedure( Func()
   let(()
      printf("In Func\n")
   )
)

when(simSimulator == "verilog"
   hnlRegPostNetlistTrigger('Func)
)

hnlDeRegPostNetlistTrigger('Func)
=> t
```

HNL Net-Based Netlisting Functions

The following HNL functions are available for use by the formatter for net-based netlisting:

hnlDoNetBased

```
hnlDoNetBased( cellList )
```

Description

The driver for net-based netlists. cellList is the list of cellviews (macros) to generate the netlist. By default, this procedure is called by `hnlPrintNetlist()` with hnlListOfAllCells as the list of cells to be netlisted.

hnlPrintsignal

```
hnlPrintSignal( )
```

Description

The procedure that finds and directs the netlisting of all signals in hnlCurrentCell. It processes each signal and sets up the needed variables for use later. It is responsible for calling the procedures `hnlPrintSignalHeader()`, `hnlPrintInst()`, and `hnlPrintSignalFooter()`.

hnlGetMasterCells

```
hnlGetMasterCells( instlist )
```

Description

Given a list of instances, this procedure returns a list of the cellviews of the instances' view-switched masters, which are stored in the same order as the given list.

hnlCloseMasterList

```
hnlCloseMasterList(inlists)
```

Description

Given a list of cellviews, this procedure closes them one by one. This is the reverse function of `hnlGetMasterCells()`.

hnlFindAllInstInCell

```
hnlFindAllInstInCell( cellview )
```

Description

Given a cellview, finds all netlistable instances in it.

hnlIsCellNetlisttable

```
hnlIsCellNetlistable( cellview )
```

Description

Given a cellview, determines if instantiation of this cellview is netlistable to HNL. This procedure will return nil for a cellview that contains no connectivity and for a cellview that has the property nlAction, with value ignore.

hnlGetTermNameOfSig

```
hnlGetTermNameOfSig( sig )
```

Description

Given a signal, finds the names of all the terminals the signal connects to. The names are the single-bit names of the bits of the terminals that connect to the signal.

hnlGetTermByName

```
hnlGetTermByName(cellview termname)
```

Description

Given a cellview and a terminal name, returns the terminal ID and index corresponding to the member termname found in the cellview specified, where the ID is the first element of the list

and the index is the second. If the given terminal name implies a width greater than 1, a -1 is returned as the index. If a single bit name is given, then the index returned is the index to the terminal.

3

VHDL Toolbox Functions

VHDL Toolbox is an integrated netlisting and simulation environment that you can use to generate structural VHDL (IEEE93/87) text netlists from hierarchical OpenAccess 2.2 schematics and run simulations in the Cadence NC environment. You can run VHDL netlister in standalone mode using Cadence Simulation Environment (SI) of Open Simulation System (OSS). For details, see the *Virtuoso VHDL Toolbox User Guide*.

This chapter describes the SKILL functions associated with VHDL Toolbox:

- VHDL Functions
- VHDL AMS Functions

VHDL Functions

This chapter describes the public SKILL functions provided by the VHDL interface:

vosHiDisplayNetlist

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

Description

Displays the generated VHDL netlist. This function can be used only when a single netlist is generated for the complete design.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

Example

```
vosHiDisplayNetlist()
```

vosLaunchIrunSimulation

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

Description

Launches the xrun simulator from the command line. If the `vhdlSimNetlistandSimulate` variable is set to `t`, the function netlists the design before launching the xrun simulator. Otherwise, it launches the simulator without netlisting the design.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

Example

```
si <runDirName> -batch -command vosLaunchIrunSimulation
```

vhdlImport

```
vhdlImport(
     t_libName
     l_srcFiles
     t_logName
     l_params
     [ g_runInBackground ]
     [ g_displayResults ]
     )
     => nil / t
```

Description

Runs vhdlin to import a list of VHDL source files into the specified library with the given parameters. The parameters are the names of the vhdlin parameters, passed in as a disembodied property list. Optionally, it can run vhdlin as a background process and/or display the results interactively.

Argument

t_libName	Name of the target library where files are imported.
l_srcFiles	List of the VHDL text files to be imported by vhdlin.
t_logName	Name of the log file for vhdlin to generate.
l_params	Disembodied Property List (DPL) defining the vhdlin parameters, where the members of the DPL match the names of the parameters used by vhdlin.
g_runInBackground	Boolean flag, specifying whether vhdlin should be run in the foreground, blocking the current session, or as a background process. Default: `nil` (runs in foreground.)
g_displayResults	Boolean flag, specifying whether the results of the vhdlin run should be displayed interactively using the VHDL Toolbox log/error viewing window. Default: `nil` (does not display results interactively.)

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

vhdlHiImport

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

Description

Builds and displays the VHDL Import form.

Arguments

None

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

vhdlHiInvokeToolBox

```
vhdlHiInvokeToolBox(
     [ t_position ]
     )
     => t / nil
```

Description

Invokes the vhdl toolbox window.

Arguments

t_position	The argument position specifies the initial position of toolbox on the screen. This value can be a SKILL list of any two integer elements. If you do not specify any values in this argument it takes ‘(0 0) as the default value for position.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

Examples

Displays the toolbox window at the x co-ordinate -9 and y co-ordinate 162 on the screen.

```
vhdlHiInvokeToolBox ('(-9 162))
```

Displays the toolbox window at the default location i.e. ‘(0 0).

```
vhdlHiInvokeToolBox()
```

vhdlToPinList

```
vhdlToPinList(
     t_libName
     t_cellName
     t_viewName
     )
     => list / nil
```

Description

Translates a VHDL cellView into an intermediate pin list format.

Arguments

t_libName	The name of the library containing the VHDL cellView.
t_cellName	The cell name of the VHDL cellView to translate.
t_viewName	The view name of the VHDL cellView to create.

Value Returned

list	Returns the pinlist if successful.
nil	Returns nil if the command is not successful.

Note the following for the returned pin list:

l_pinList	A DPL list describing the cellview ports, cellview properties, and port properties in the following format:
<l_pinList>	(nil ports <portList> [props <propList>]
<portList>	(<port> [<portList>])
<port>	(nil name "termName" direction "termDir"
	[prop <propList>] [pins <pinList>])
<propList>	(<prop> [<propList>])
<prop>	(nil s_propName t_propValue s_propName t_propValue)
<pinList>	(<pin> [<pinList>])

`<pin>`	(nil name "pinName" accessDir "pinAccessDir" [prop <propList>])

Example

```
pinList =
     list(nil
          'ports list(
               list(nil 'name "clock" 'direction "input")
               list(nil 'name "a0"    'direction "output"
                    'prop list(nil 'delay 55.0))
          )
          'prop list(nil 'partName "count" 'myProp 5)
     )
```

vhdlPinListToVHDL

```
vhdlPinListToVHDL(
     t_libName
     t_cellName
     t_viewName
     l_pinList
     )
     => t / nil
```

Description

Allows the generation of VHDL views from an intermediate pin list format.

Arguments

t_libName	The name of the library to store the VHDL cellView.
t_cellName	The cell name of the translated VHDL cellView.
t_viewName	The view name of the translated VHDL cellView.
l_pinList	The pin list to be translated to the VHDL cellView

The pinList has the following format:

```
( ports (nil name "termName" direction "termDir"
    pins ( (name "pinName" accessDir "pinAccessDir"
            pinPropName pinPropValue ...)
            .   .   .   .   .   .   .   .
            (name "pinName" accessDir "pinAccessDir"
               pinPropName pinPropValue ...)

                prop (termPropName termPropValue ...)

            .   .   .   .   .   .   .   .   .   .   .   .   .
          .   .   .   .   .   .   .   .   .   .   .   .   .
          .   .   .   .   .   .   .   .   .   .   .   .   .
          (nil name "termName" direction "termDir"
             pins ( (name "pinName" accessDir "pinAccessDir"
                      pinPropName pinPropValue ...)
                       .   .   .   .   .   .   .   .
                      (name "pinName" accessDir "pinAccessDir"
                      pinPropName pinPropValue ...)

             prop (termPropName termPropValue ...)

          prop (nil cvPropName cvPropValue ...)

    Returns
      t if successful
```

```
        else nil
        )
```

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

vhdlRegisterSimulator

```
vhdlRegisterSimulator(
     [ parserCallBack ]
     [ analyzerCallBack ]
     [ analyzerFileExt ]
     [ elaboratorCallBack ]
     [ simulatorCallBack ]
     [ dataDirCallBack ]
     [ dataFileCallBack ]
     [ workLibCallBack ]
     )
     => t / nil
```

Description

To use non-Cadence VHDL tools, you need to define your own SKILL procedures and register this information with the toolbox using the SKILL routine, `vhdlRegisterSimulator()`.

To register your callbacks, add the procedures for the callbacks in some file, say myfile.il that is in the /home/xyz directory and add the following lines to the .cdsinit file in your home directory:

```
(loadi "/home/xyz/myfile.il")
```

If you do not provide your own callback routines to invoke any of the non-Cadence tools, namely, the `parser/analyser/elaborator/simulator`, then by default, XM-VHDL tools such as the `parser/analyzer xmvhdl`, elaborator `xmelab` and simulator `xmsim` are run.

Arguments

parserCallBack	This procedure takes the VHDL source file and the name of the library in which this file is contained and runs the parser on it.
analyzerCallBack	The procedure invokes the analyzer that analyzes the specified *sourceFileName* which exists in the specified directory `filePath`.
analyzerFileExt	It is a string representing the name of the analyzed file.
elaboratorCallBack	This procedure invokes the elaborator to elaborate the VHDL design unit.
simulatorCallBack	The procedure invokes the simulator that simulates the specified simulation model.

dataDirCallBack	Given the library, cell, and view name this procedure returns the physical directory where the VHDL text file is to be stored.
dataFileCallBack	Given the library, cell and view names this procedure. returns the physical file name under which the VHDL text file is to be stored.
workLibCallBack	This procedure returns the library that contains the compiled design unit information.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

VHDL AMS Functions

This section describes the VHDL AMS SKILL functions.

vhmsCompilationFailure

```
vhmsCompilationFailure(
     t_libName
     t_cellName
     t_viewName
     fileName
     )
     => list / nil
```

Description

Informs you about the compilation failures of the file specified with the *fileName* parameter. A compilation error message is displayed and you can open the file for editing by clicking 'Yes'.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.
t_viewName	The name of the view you are working on.
fileName	The name of the file to be compiled.

Value Returned

`list`	List of compilation failures.
`nil`	Returns `nil` if not successful.

vhmsDefaultEdit

```
vhmsDefaultEdit(
     t_libName
     t_cellName
     t_viewName
     b_fileObj
     t_mode
     [ errFile ]
     )
     => t / nil
```

Description

Opens an existing file or creates a template and opens it in an editor. If the mode is `r` (read only) then it tries to open an existing file.

First an entity needs to be created, only then can the architecture, package, or body views be created. When creating a template for an entity, if the symbol exists, then this function calls the `vhmsSymbolToPinListGen()` function and then use this pinlist to create the template.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.
t_viewName	The name of the view you are working on.
b_fileObj	The file object obtained from the ddGetObj() procedure.
t_mode	Read or write mode.
errFile	Name of the error file. This is an optional parameter.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

vhmsSaveFile

```
vhmsSaveFile(
     t_libName
     t_cellName
     t_viewName
     t_fileName
     )
     => list / nil
```

Description

Compiles the vhms file. If the symbol does not exist, then it prompts you to create the symbol. If the compilation fails then it calls the `vhmsCompilationFailure` function.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.
t_viewName	The name of the view you are working on.
t_fileName	The vhms file name.

Value Returned

list	The created vhms compilation.
nil	Returns `nil` if not successful.

vhmsGetCellParameters

```
vhmsGetCellParameters(
     t_libName
     t_cellName
     )
     => list / nil
```

Description

Returns the CDF parameters of the vhdl generics. The sub-list returned contains the name of the generic, the default value, and the type.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.

Value Returned

list	Returns the CDF parameters of the vhdl generics.
nil	Returns nil if ddGetObj is unable to return a valid lib/cell/view name, or if there are no CDF parameters defined, or if it is not able to read the CDF.

vhmsPinListToVHDLAMS

```
vhmsPinListToVHDLAMS(
     t_libName
     t_cellName
     t_viewName
     l_pinList
     )
     => t / nil
```

Description

Creates a vhdlams entity, package, body, configuration, or architecture depending on the view name. Specify the view name as ‘entity’, ‘package’, ‘body’, or ‘configuration’ for creating an entity, package, body, or configuration respectively. If nothing is specified, then an architecture view is created. The view is created in `./libName/cellName/viewName`.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.
t_viewName	The name of the view you are working on.
l_pinList	The pin list to be converted.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` if not successful.

vhmsSymbolToPinListGen

```
vhmsSymbolToPinListGen(
     t_libName
     t_cellName
     [ viewName ]
     )
     => l_pinList / nil
```

Description

Generates a pin list from the symbol.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.
viewName	The view name. This is an optional parameter and its default value is symbol.

Values Returned

l_pinList	The generated pin list.
nil	If unsuccessful. `vhmsSymbolToPinListGen` fails only when `schSymbolToPinList` is not callable.

vhmsToPinList

```
vhmsToPinList(
     t_libName
     t_cellName
     t_viewName
     )
     => list / nil
```

Description

Translates a VHDLAMS cellView into an intermediate pin list format. The pin list represents all the ports to the VHDLAMS description and their directions, such as ‘input’, ‘output’, and ‘inout’.

Arguments

t_libName	The name of the library you are working on.
t_cellName	The name of the cell you are working on.
t_viewName	The name of the view you are working on.

Value Returned

list	Returns the pin list.
nil	Returns nil if not successful.

vhmsUpdateCellCDFParams

```
vhmsUpdateCellCDFParams(
     d_cellId
     l_pinList
     )
     => cdf_obj
```

Description

Updates the CDF parameters. It reads all the parameters from the pin list. If a given parameter does not exist in the CDS, then it updates the CDF.

Arguments

d_cellId	The cellView identifier.
l_pinList	The pin list for which the CDF parameters need to be updated.

Value Returned

cdf_obj	The CDF object.